How Branch Mispredictions Affect Quicksort

Kanela Kaligosi[1] and Peter Sanders[2]

[1] Max Planck Institut für Informatik
Saarbrücken, Germany
kaligosi@mpi-sb.mpg.de
[2] Universität Karlsruhe, Germany
sanders@ira.uka.de

Abstract. We explain the counterintuitive observation that finding "good" pivots (close to the median of the array to be partitioned) may not improve performance of quicksort. Indeed, an intentionally *skewed* pivot *improves* performance. The reason is that while the instruction count decreases with the quality of the pivot, the likelihood that the direction of a branch is mispredicted also goes up. We analyze the effect of simple branch prediction schemes and measure the effects on real hardware.

1 Introduction

Sorting is one of the most important algorithmic problems both practically and theoretically. Quicksort [1] is perhaps the most frequently used sorting algorithm since it is very fast in practice, needs almost no additional memory, and makes no assumptions on the distribution of the input. Hence, quicksort, its analysis and efficient implementation is discussed in most basic courses on algorithms. When we take a random pivot, the expected number of comparisons is $2n \ln n \approx 1.4n \lg n$. One of the most well known optimizations is that taking the median of three elements reduces the expected number of comparisons to $\frac{12}{7} n \ln n \approx 1.2n \lg n$ [2]. Indeed, by using the median of a larger random sample, the expected number of comparisons can be made as close to $n \lg n$ as we want [3]. For sufficiently large inputs, the increased overhead for pivot selection is negligible. At first glance, counting comparisons makes a lot of practical sense since in quicksort, the number of executed instructions and cache faults grow proportionally with this figure.

However, in comparison based sorting algorithms like quicksort or mergesort, neither the executed instructions nor the cache faults dominate execution time. Comparisons are much more important, but only indirectly since they cause the direction of branch instructions depending on them to be mispredicted. In modern processors with long execution pipelines and superscalar execution, dozens of subsequent instructions are executed in parallel to achieve a high peak throughput. When a branch is mispredicted, much of the work already done on the instructions following the predicted branch direction turns out to be wasted. Therefore, ingenious and very successful schemes have been devised to accurately *predict* the direction a branch takes. Unfortunately, we are facing a

Y. Azar and T. Erlebach (Eds.): ESA 2006, LNCS 4168, pp. 780–791, 2006.

dilemma here. Information theory tells us that the optimal number of $\approx n \lg n$ element comparisons for sorting can only be achieved if each element comparison yields one bit of information, i.e., there is a 50 % chance for the branch to take either direction. In this situation, even the most clever branch prediction algorithm is helpless. A painfully large number of branch mispredictions seems to be unavoidable.

Related Work: This dilemma can be circumvented by devising sorting algorithms where comparisons are decoupled from branches [4]. However, the algorithm proposed in [4] is not in-place and requires compiler optimizations that are not universally available yet. Hence it remains interesting to see what can be done about branch mispredictions in quicksort. [5] based on a discussion between Sanders and Moruz in 2004 observes that a reduced number of branch mispredictions improves the running time of quicksort when inputs are almost sorted. In [6], a variant of multiway mergesort is proposed that reduces branch mispredictions by sequentially searching for the next element to be merged. This algorithm is analyzed for the case of static branch prediction. Compared to this, the innovation of the present paper is that it gives experimental results and concerns a classical, in-place algorithm. Moreover, for quicksort also dynamic branch prediction is interesting. Martinez and Roura [3] note that nonmedian pivots can be beneficial if swaps are much more expensive than comparisons. However, it seems that this situation would correspond to a nonoptimal use of quicksort because then it would be more efficient to sort references to the elements first, followed by a permutation of the original input.

Overview: In Section 2, we review quicksort and basic branch prediction mechanisms. Section 3 outlines our main theoretical contributions — an analysis of quicksort in the context of branch mispredictions. For simplicity we assume that the elements are distinct. We look at two variants of quicksort: random and skewed pivot, and three branch prediction methods: static, 1-bit predictor and 2-bit predictor. To the best of our knowledge this represents the first analysis of the interactions of a nontrivial algorithm with dynamic branch prediction methods. Note that static branch prediction is not useful for analyzing quicksort variants like random pivot that try to approximate the median. The theoretical results are complemented by experiments in Section 4. In particular, there we also look at the classical median-of-three pivot selection. It turns out that this frequently used improvement only gives a negligible advantage over random pivot. Its advantages wrt. instruction count basically cancel with its disadvantages wrt. branch prediction. Somewhat surprisingly, taking a pivot with rank around $n/10$ can lead to a better performance.

2 Preliminaries

In this section we give a more detailed description of quicksort and then give an overview of several branch prediction schemes.

2.1 Quicksort

A simple pseudocode of quicksort sufficient for our purposes can be seen in Algorithm 1. In the rest of the paper, it will be clear from the context, whether n denotes the input size or the currently relevant subproblem size. The algorithm can be instantiated with different subroutines for determining the pivot. We distinguish between three basic schemes: *random pivot*, *median-of-three* random elements, and α-*skewed pivot*, i.e., a median of rank αn. Note that the latter scheme is an idealization because in practice only approximations of this value can be obtained efficiently (using random sampling [3]). However, for sufficiently big inputs, one could get very good approximations at negligible cost for all but the lowest levels of recursion.

Algorithm 1. Sort array part $a[\ell..r]$

```
Procedure quicksort(ℓ, r : integer);
  if r > ℓ then
    i = ℓ; j = r; x = pivot();
    repeat
      while a[i] < x do i++ ; endwhile {Loop I}
      while a[j] > x do j−− ; endwhile {Loop J}
      if i ≤ j then swap(a[i], a[j]);
    until j ≤ i
    quicksort(ℓ, i − 1);
    quicksort(i + 1, r);
  end if
```

2.2 Branch Prediction Schemes

In *static branch prediction* the compiler once and for all labels a branch instruction as predict-taken or as predict-not-taken. This scheme does not take into account the dynamic behavior of the program. Static prediction is useful together with α-skewed pivot selection. For $\alpha < 1/2$, the compiler should statically predict that Loop I is not executed and that Loop J is executed.[1]

In the standard versions of pivot selection that attempt to approximate the median, static prediction does not help. Here *dynamic branch prediction* mechanisms provided by the hardware may do better.

The simplest dynamic scheme is a *1-bit predictor*. The hardware always predicts a branch instruction to take the same direction it took the last time it was executed.

A refined version working better in practice is the *2-bit predictor*. In order to further improve the prediction accuracy, 2-bit prediction schemes were introduced. In these schemes the prediction must be wrong twice before it is changed. See for example [7]. In Fig. 1 we can see the behavior of a 2-bit predictor scheme.

In fact we can have the general case of a k-bit counter. As in the 2-bit case, the counter is incremented if the branch is taken and decremented if the branch

[1] Modern compilers do that using profiling information.

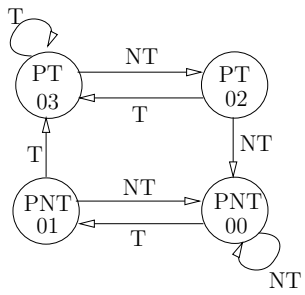

Fig. 1. 2-bit prediction scheme: There are four states, where PT means Predict Taken and PNT means Predict Not Taken. The arrows show how the states are changed when a branch is taken T or not taken NT.

is not taken. The branch is predicted taken when the counter is greater than or equal to half of its maximum value and not taken otherwise. The k-bit prediction schemes are not widely used since studies have shown that the 2-bit prediction scheme is good enough for all practical purposes.

Furthermore, there are branch prediction schemes which not only consider the history of the particular branch to be predicted but also that of other branches which may be related to the current branch and affect its outcome. In this way the prediction accuracy is further improved. See [7] for more details. It looks difficult to analyze quicksort for the most general schemes. It also seems that simple local prediction is adequate in the case of quicksort since the past behavior of a branch instruction is likely to yield information whether the pivot is larger or smaller than the median.

3 Analysis

In this section we analyze the behavior of quicksort in terms of the number of branch mispredictions it incurs. We give the analysis of the branch mispredictions occurring in the two inner and consecutive while loops of quicksort that perform the partitioning step. Note that the remaining branch instructions are much less frequently executed or easy to predict.

In the next three subsections we outline the proof of the following theorem.

Theorem 1. *Let* $H(\alpha) = -(\alpha \lg(\alpha) + (1-\alpha)\lg(1-\alpha))$ *be the binary entropy function. The number of branch mispredictions that occur during the execution of the partitioning step of quicksort are as described in Table 1. The entries for the 1-bit and the 2-bit predictor give the expected number of branch mispredictions given the assumption that there is a probability* α *of an element being smaller when compared with the pivot that has rank* αn.[2] *For the entry random pivot*

[2] This assumption means that our analysis is "heuristic" since the knowledge that the pivot has rank αn introduces slight dependencies between the comparisons. It is an interesting question whether there is an easy argument proving the same bounds for the standard average case model.

Table 1. Number of branch mispredictions

	random pivot	α-skewed pivot
static predictor	$\frac{\ln 2}{2} n \lg n + \mathcal{O}(n)$, $\frac{\ln 2}{2} \approx 0.3466$	$\frac{\alpha}{H(\alpha)} n \lg n + \mathcal{O}(n)$, $\alpha < 1/2$ $\frac{1-\alpha}{H(\alpha)} n \lg n + \mathcal{O}(n)$, $\alpha \geq 1/2$
1-bit predictor	$\frac{2 \ln 2}{3} n \lg n + \mathcal{O}(n)$, $\frac{2 \ln 2}{3} \approx 0.4621$	$\frac{2\alpha(1-\alpha)}{H(\alpha)} n \lg n + \mathcal{O}(n)$
2-bit predictor	$\frac{28 \ln 2}{45} n \lg n + \mathcal{O}(n)$, $\frac{28 \ln 2}{45} \approx 0.4313$	$\frac{2\alpha^4 - 4\alpha^3 + \alpha^2 + \alpha}{(1-\alpha(1-\alpha))H(\alpha)} n \lg n + \mathcal{O}(n)$

with static predictor there is no such assumption and for the entry α-skewed with static predictor we give a worst case analysis.

In Fig. 2 we see the α-dependent coefficients of $n \lg n$ for the case of the α-skewed pivot. As expected they are maximized for $\alpha = 0.5$ and their value decreases as we move towards smaller or larger α's. Moreover, the best curve is the one for the static predictor, followed by the one for the 2-bit predictor and then the one for the 1-bit predictor.

3.1 Static Prediction Scheme

Next we analyze the number of branch mispredictions quicksort could achieve with static branch prediction if somebody would tell the predictor whether the pivot is smaller or larger than the median. We can judge dynamic branch prediction by comparing its performance with this "best possible" prediction. We consider the random pivot and the α-skewed pivot case. For the former we give

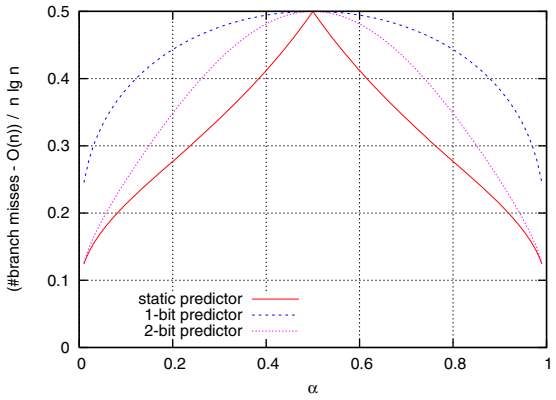

Fig. 2. The α-dependent coefficients of $n \lg n$ for varying α

an expected case analysis that holds for every input, namely we make no assumptions for the distribution of the input. For the latter we give a worst case analysis.

Let $B^{\text{stat}}(n)$ denote the expected number of branch mispredictions occurring in the partitioning step of quicksort with random pivot when a static predictor is used. Consider one execution of the partitioning step. Let x be the pivot element and αn its rank for some $0 < \alpha \leq 1$. The rank αn of the pivot can take each of the values $1, \ldots, n$ with equal probability and after the partitioning we are left with subproblems of size $\alpha n - 1$ and $(1-\alpha)n$. If $\alpha < 1/2$ then each element smaller than the pivot causes a branch misprediction, since Loop I is predicted not to be executed and Loop J is predicted to be executed. Therefore, we have at most αn branch mispredictions. If $\alpha \geq 1/2$ the prediction of the loop is the other way around and each element larger than the pivot causes a branch misprediction and therefore we have $(1-\alpha)n$ mispredictions. So, we set up the following recurrence for $n \geq 1$, with $B^{\text{stat}}(0) = 1$.

$$B^{\text{stat}}(n) \leq \frac{1}{n}\Big(\sum_{\alpha n=1}^{\lfloor n/2 \rfloor} B^{\text{stat}}(\alpha n - 1) + B^{\text{stat}}((1-\alpha)n) + \alpha n) + \sum_{\alpha n=\lfloor n/2 \rfloor+1}^{n} (B^{\text{stat}}(\alpha n - 1) + B^{\text{stat}}((1-\alpha)n) + (1-\alpha)n\Big).$$

We solve the recurrence using for example the technique in [8] and we obtain $B^{\text{stat}}(n) \leq \frac{\ln 2}{2} n \lg n + \mathcal{O}(n)$.

Now let $A^{\text{stat}}(n)$ be the number of branch mispredictions of quicksort with α-skewed pivot when the static predictor is used. Similarly to above if $\alpha < 1/2$ each element smaller than the pivot causes a branch misprediction and if $\alpha \geq 1/2$ each element larger than the pivot causes a branch misprediction. So, we set up the following recurrence for $n \geq 1$, with $A^{\text{stat}}(0) = 1$.
$A^{\text{stat}}(n) \leq \alpha n + A^{\text{stat}}(\alpha n - 1) + A^{\text{stat}}((1-\alpha)n)$, if $\alpha < 1/2$ and
$A^{\text{stat}}(n) \leq (1-\alpha)n + A^{\text{stat}}(\alpha n - 1) + A^{\text{stat}}((1-\alpha)n)$, if $\alpha \geq 1/2$. We can prove by induction that $A^{stat}(n) \leq \frac{\alpha}{H(\alpha)} n \lg n + \mathcal{O}(n)$, if $\alpha < 1/2$ and
$A^{stat}(n) \leq \frac{1-\alpha}{H(\alpha)} n \lg n + \mathcal{O}(n)$, if $\alpha \geq 1/2$.

3.2 1-Bit Prediction Scheme

We now analyse quicksort when a 1-bit prediction scheme is used. In the 1-bit prediction scheme we predict that a branch instruction will go in the same direction as the last time it was executed. Let X_i be the indicator random variable which is 1 if the i-th element in Loop I causes a branch misprediction and 0 otherwise. Correspondingly we define Y_j for Loop J. We have that $X_i = 1$ if $a[i] \geq x$ and $a[i-1] < x$ or if $a[i] < x$ and $a[i-1] \geq x$. Using our assumption that $P[a[i] < x] = \alpha$, we get $P[X_i = 1] = 2\alpha(1-\alpha)$. Similarly $P[Y_j = 1] = 2\alpha(1-\alpha)$. Let $X = \sum_{i=1}^{k} X_i + \sum_{j=k+1}^{n} Y_j$ denote the number of mispredictions. Then $E[X] = E[\sum_{i=1}^{k} X_i + \sum_{j=k+1}^{n} Y_j] = \sum_{i=1}^{n} E[X_i] = nP[X_1 = 1] = 2\alpha(1-\alpha)n$.

Let $B^{\text{1-bit}}(n)$ denote the expected number of branch mispredictions when random pivot is used. Then we obtain the recurrence
$B^{\text{1-bit}}(n) \leq \frac{1}{n}\sum_{\alpha n=1}^{n}\Big(B^{\text{1-bit}}(\alpha n-1)+B^{\text{1-bit}}((1-\alpha)n)+2\alpha(1-\alpha)n\Big)$.
This solves to $B^{\text{1-bit}}(n)=\frac{2\ln 2}{3}n\lg n+\mathcal{O}(n)$.

Now let $A^{\text{1-bit}}(n)$ denote the expected number of branch mispredictions when an α-skewed pivot is used. Then
$A^{\text{1-bit}}(n) \leq 2\alpha(1-\alpha)n+A^{\text{1-bit}}(\alpha n-1)+A^{\text{1-bit}}((1-\alpha)n)$.
It can be shown by induction that it solves to $A^{\text{1-bit}}(n)=\frac{2\alpha(1-\alpha)}{H(\alpha)}+\mathcal{O}(n)$.

3.3 2-Bit Prediction Scheme

We now consider the 2-bit prediction scheme. As stated earlier we assume that an element is smaller than the pivot with probability α independently of the other comparisons. With this simplification, the branch predictor can be modeled as a Markov chain. First consider the predictor of Loop I. Its corresponding Markov chain has four states, each one corresponding to a state of the predictors automaton, see Fig. 1. The transition table where entry P_{kl} represents the probability of going to state l given that we are in state k is as follows.

$$\mathbf{P}=\begin{bmatrix}\alpha & 1-\alpha & 0 & 0\\ \alpha & 0 & 0 & 1-\alpha\\ \alpha & 0 & 0 & 1-\alpha\\ 0 & 0 & \alpha & 1-\alpha\end{bmatrix}$$

Let π_0,π_1,π_2,π_3 denote the stationary probabilities of the Markov chain, i.e., they are the solution to the system $\overrightarrow{\pi}\cdot\mathbf{P}=\overrightarrow{\pi}$ and $\sum_{k=0}^{3}\pi_i=1$. Then $\pi_0=\frac{\alpha^2}{1-\alpha(1-\alpha)}$, $\pi_1=\frac{\alpha^2(1-\alpha)}{1-\alpha(1-\alpha)}$, $\pi_2=\frac{\alpha(1-\alpha)^2}{1-\alpha(1-\alpha)}$ and $\pi_3=\frac{(1-\alpha)^2}{1-\alpha(1-\alpha)}$. One can easily verify this by substitution. Similarly, to the above we obtain the Markov chain corresponding to Loop J. Now, let X_i be the indicator random variable which is 1 if the i-th element of the Loop I causes a branch misprediction and 0 otherwise. Correspondingly we define Y_j for Loop J.

The ith element causes a branch misprediction in the following cases. After having considered element $a[i-1]$ the Markov chain is in state 0 and $a[i]\geq x$, or it is in state 1 and $a[i]\geq x$, or it is in state 2 and $a[i]<x$ or it is in state 3 and $a[i]<x$. Therefore,

$$P[X_i=1]=\pi_0\cdot P[a[i]\geq x]+\pi_1\cdot P[a[i]\geq x]+\pi_2\cdot P[a[i]<x]+\pi_3\cdot P[a[i]<x].$$

By substituting $P[a[i]\geq x]=1-\alpha$ and $P[a[i]<x]=\alpha$ and the values for $\pi_1,\ldots,\pi_3$ we obtain that $P[X_i=1]=\frac{2\alpha^4-4\alpha^3+\alpha^2+\alpha}{1-\alpha(1-\alpha)}$. The same holds for $P[Y_i=1]$. Now let $X=\sum_{i=1}^{k}X_i+\sum_{j=k+1}^{n}Y_j$ be the number of branch mispredictions. Then $E[X]=E[\sum_{i=1}^{k}X_i+\sum_{j=k+1}^{n}Y_j]=\sum_{i=1}^{n}E[X_i]=nP[X_1=1]=\frac{2\alpha^4-4\alpha^3+\alpha^2+\alpha}{1-\alpha(1-\alpha)}n$. Let $B^{\text{2-bit}}(n)$ denote the expected number of branch mispredictions of quicksort with random pivot. Then

$B^{2\text{-bit}}(n) \leq \frac{1}{n}\sum_{\alpha n=1}^{n}\Big(B^{2\text{-bit}}(\alpha n-1)+B^{2\text{-bit}}((1-\alpha)n)+\frac{2\alpha^4-4\alpha^3+\alpha^2+\alpha}{1-\alpha(1-\alpha)}n\Big)$. This solves to $B^{2\text{-bit}}(n) = \frac{28\ln 2}{45} n\lg n + \mathcal{O}(n)$.

Now let $A^{2\text{-bit}}(n)$ be the expected number of branch mispredictions of quicksort with α-skewed pivot. Then
$A^{2\text{-bit}}(n) \leq \frac{2\alpha^4-4\alpha^3+\alpha^2+\alpha}{1-\alpha(1-\alpha)}n + A^{2\text{-bit}}(\alpha n-1) + A^{2\text{-bit}}((1-\alpha)n)$, which solves to
$A^{2\text{-bit}}(n) = \frac{2\alpha^4-4\alpha^3+\alpha^2+\alpha}{(1-\alpha(1-\alpha))H(\alpha)} n\lg n + \mathcal{O}(n)$.

4 Experiments

For our experiments we use one of the fastest quicksort implementations std::sort from the STL library included in GCC v3.3. This implementation uses the median of 3 elements as the pivot. We added an implementation of the random pivot and the idealized α-skewed pivot mechanisms. Our inputs are random permutation of the integers in the range $[1,\ldots,n]$. We average over $\max\{100, \lceil 10^7/n\rceil\}$ inputs. Note that with a simple calculation we can obtain the element of rank αn, for a given α. Observe that this makes the cost of finding the pivot element negligible. If the time taken by quicksort is too large, the STL implementation switches to an algorithm of $\mathcal{O}(n\lg n)$ worst case performance. Since we are only interested in quicksort we have removed this switch. In order to be able to use a larger number of α's for the skewed pivot mechanism we changed the threshold of breaking the recursion from 16 to 20 elements. This does not have any significant effects. The STL implementation uses insertion sort for sorting the small instances. The measures in our figures include the cost of the final insertion sort. This changes the cost of all algorithms by the same amount and therefore does not affect our conclusions.

We used the PAPI tool which provides an interface that allows to count several CPU events including the number of branch mispredictions and the number of instructions executed. When not otherwise stated, the experiments are on a 3GHz Pentium 4 Prescott.

Figs. 3, 4 and 5 show a comparison of the random pivot, the median of 3, the exact median or 1/2-skewed and the 1/10-skewed pivoting mechanisms in terms of the execution time, the number of occurring branch mispredictions and the number of instructions executed for different values of n. In Fig. 3 we see that the random pivot algorithm is most of the times a little bit worse than the others. On the other hand the difference is very small and in particular we observe that the curves for the random pivot, the median of 3 and the exact median are very close to each other, in contrast to the common concept that the exact median and the median of 3 should significantly outperform the random pivot. Furthermore, we see that the 1/10-skewed algorithm has a better performance.

In Fig. 4 we see that the random pivot has for most n a smaller number of branch mispredictions compared to the median of 3 and the exact median. The measured prediction quality is better than the quality we would expect for a 1-bit predictor (see Table 1) but not quite as good as to be expected for a 2-bit predictor. The 1/10-skewed pivot algorithm has of course the smallest number

of branch mispredictions. In Fig. 5 we see the number of instructions that are executed. These are proportional to the number of comparisons and therefore we see that the exact median is the best, followed by the median of 3, then the random pivot and finally the 1/10-skewed pivot. Observe that the curves in this figure are very flat and smooth in contrast to the curves in Fig. 3. Therefore, it is not only the number of executed instructions that plays a major role in the running time. The fluctuations in Fig. 3 indicate architectural effects. Observe that for $n = 2^{16}$ the number of branch mispredictions of random pivot drop and for this n we also see a significant drop in its running time. Having a closer look at the curves we see that the curves of time and those of the branch mispredictions have the same shape, in the sense that when the branch mispredictions drop, the running time drops too and when the branch mispredictions increase the running time increases too. Note that the branch mispredictions only slowly approach

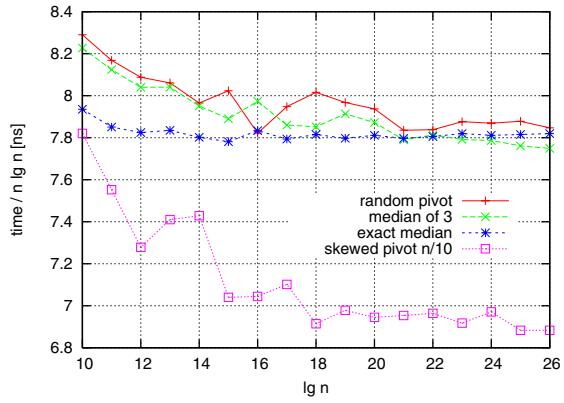

Fig. 3. Time / $n \lg n$ for random pivot, median of 3, exact median, 1/10-skewed pivot

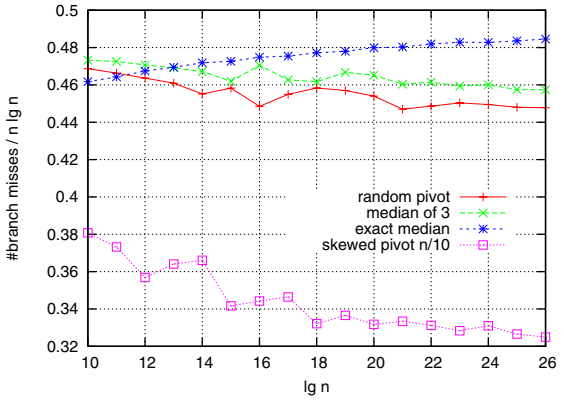

Fig. 4. Number of branch mispredictions / $n \lg n$ for random pivot, median of 3, exact median, 1/10-skewed pivot

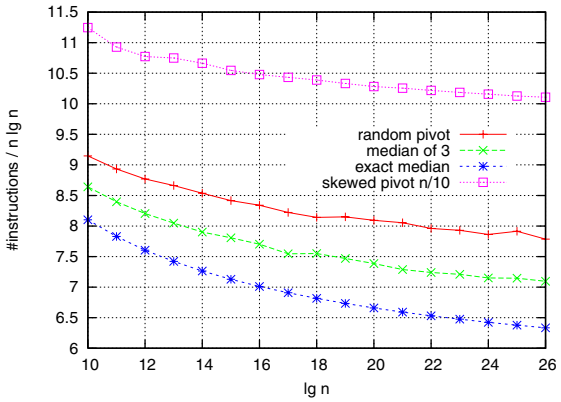

Fig. 5. Number of instructions / $n \lg n$ for random pivot, median of 3, exact median, 1/10-skewed pivot

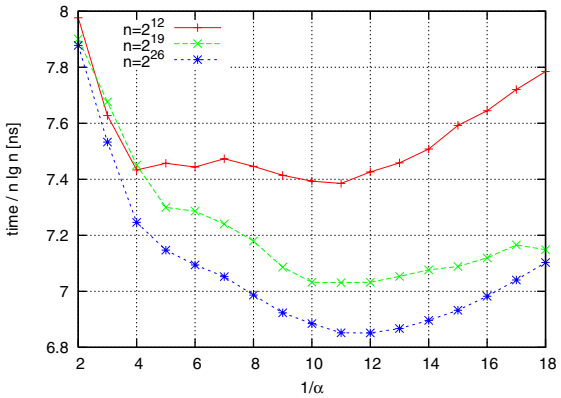

Fig. 6. Time / $n \lg n$ for different values of α

$0.5n \lg n$ for the exact median algorithm. The main reason is that the insertion sort used for small subproblems incurs only $\mathcal{O}(n)$ branch mispredictions (each iteration of the inner loop of insertion sort incurs just one branch misprediction).

Figs. 6, 7 and 8 show the performance of the α-skewed pivot when we vary α. We tried three different values of n, i.e. $2^{12}, 2^{19}$ and 2^{26}. In Fig. 6, where the running time is measured, we see that we have a parabola like figure and for $\alpha = 1/11$ we get the best running time. Moreover, the exact median which is for $\alpha = 1/2$ is a lot worse. Figs. 7 and 8 indicate why we have such a shape in Fig. 6. As α increases, the number of branch mispredictions decreases and the number of instructions increases. Therefore, we see that $\alpha = 1/11$ is the place of compromise.

In order to see the effects of different architectures we reran the experiments on an Athlon, an Opteron and a Sun machine (Figures will be in the full paper). We see for large inputs, pivots close to the median *are* an advantage. Our inter-

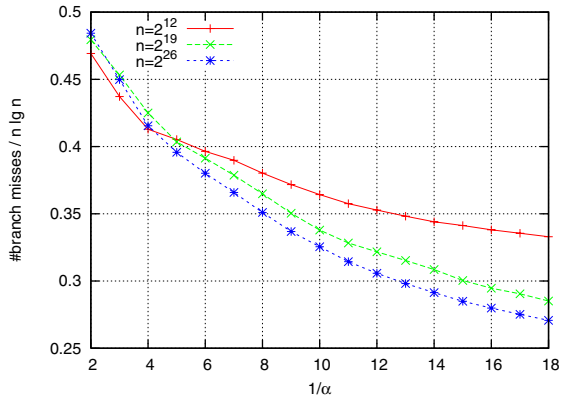

Fig. 7. Number of branch mispredictions / $n \lg n$ for different values of α

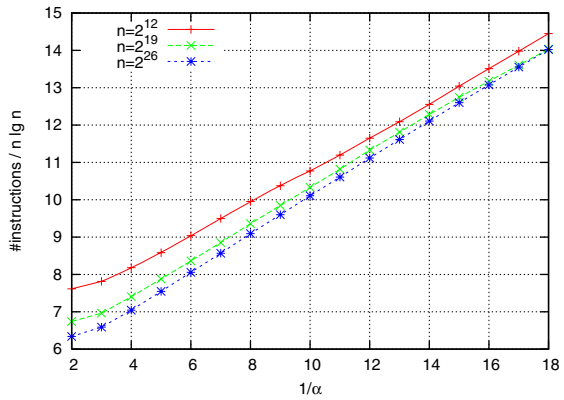

Fig. 8. Number of instructions / $n \lg n$ for different values of α

pretation is that on the Opteron, memory bandwidth is more of an issue than on the Pentium 4 architecture (perhaps its long pipelines make branch misprediction more predominant). Hence, for a skewed pivot algorithm one might want to pick α close to $1/2$ for large subproblems but use a smaller value when a subproblem fits in cache. A similar strategy might be useful on a Pentium 4, when we sort larger objects. Since our goal is understanding branch mispredictions rather than designing an efficient algorithm, we do not dwell on this issue.

5 Conclusions

Somewhat astonishingly, generally accepted "improvements" of quicksort such as median-of-three pivot selection bring no significant benefits in practice (at least for sorting small objects) because they increase the number of branch mispredictions. For teaching this means that we should either stop after random

pivots or give the full story of what happens for more sophisticated pivot selection strategies. By actively choosing a skewed pivot, we can slightly improve the performance of quicksort. Since this increases the instruction count, the better approach seems to be to avoid branch mispredictions altogether, e.g. using the techniques described in [4]. However, an in-place sorting algorithm that is better than quicksort with skewed pivots is an open problem.

Acknowledgments. We would like to thank Roman Dementiev, Dimitrios Michail and Johannes Singler for crucial assistance with the experiments.

References

1. Hoare, C.A.R.: Algorithm 64: Quicksort. Commun. ACM **4**(7) (1961) 321
2. Knuth, D.E.: The Art of Computer Programming—Sorting and Searching. Volume 3. Addison Wesley (1973)
3. Martínez, C., Roura, S.: Optimal sampling strategies in Quicksort and Quickselect. SIAM Journal on Computing **31**(3) (2002) 683–705
4. Sanders, P., Winkel, S.: Super scalar sample sort. In: 12th European Symposium on Algorithms (ESA). Volume 3221 of LNCS., Springer (2004) 784–796
5. Brodal, G.S., Fagerberg, R., Moruz, G.: On the adaptiveness of quicksort. In: Workshop on Algorithm Engineering & Experiments, SIAM (2005) 130–149
6. Brodal, G.S., Moruz, G.: Tradeoffs between branch mispredictions and comparisons for sorting algorithms. In: Proc. 9th International Workshop on Algorithms and Data Structures. Volume 3608 of Lecture Notes in Computer Science., Springer Verlag, Berlin (2005) 385–395
7. Patterson, D.A., Hennessy, J.L.: Computer Architecture: A Quantitative Approach 3rd. ed. Morgan Kaufmann (2003)
8. Sedgewick, R.: Algorithms (Second Edition). Addison-Wesley Longman Publishing Co. (1988)