On the Adaptiveness of Quicksort

Rolf Fagerberg
Dept. of Mathematics and Computer Science
University of Southern Denmark

Joint work with Gerth Stølting Brodal and Gabriel Moruz

Appeared at ALENEX'05

CAOS Seminar, ITU, Copenhagen, April 14, 2005

Quicksort

- Introduced by Hoare in 1961
- Simple, randomized sorting algorithm
- Expected number of comparisons $\sim 1.4 n \log_2 n$ [Hoare'62]
- Expected number of swaps is $1/6$ the expected number of comparisons [Hoare'62]
- In-place sorting algorithm: elements are compared and swapped within the input array (plus a runtime stack)
- In practice very fast. The all-round sorting algorithm of choice (glibc, STL, JDK, .NET).

Adaptiveness

- Adaptive sorting - the running time depends both on the input size and the presortedness in the input
- A common measure of presortedness:

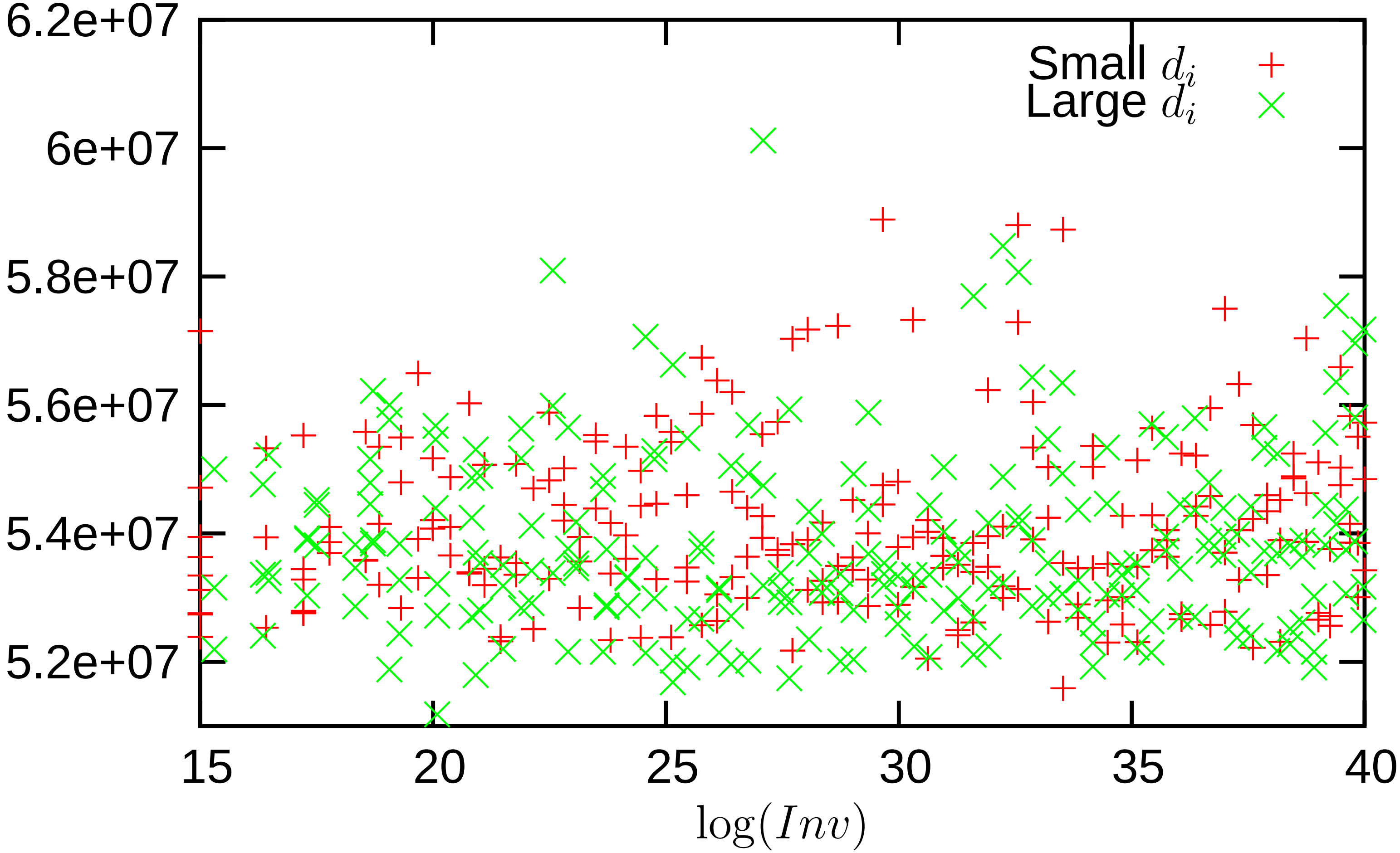
$$Inv(x_1 \dots x_n) = |\{(i,j) \mid i < j \wedge x_i > x_j\}|$$

$$Inv(1,2,3,4) = 0, Inv(4,3,2,1) = 6, Inv(2,1,4,3) = 2$$

- An optimal sorting algorithm with respect to Inv performs $\Theta(n(1 + \log(1 + \frac{Inv}{n})))$ comparisons [Manilla '85]

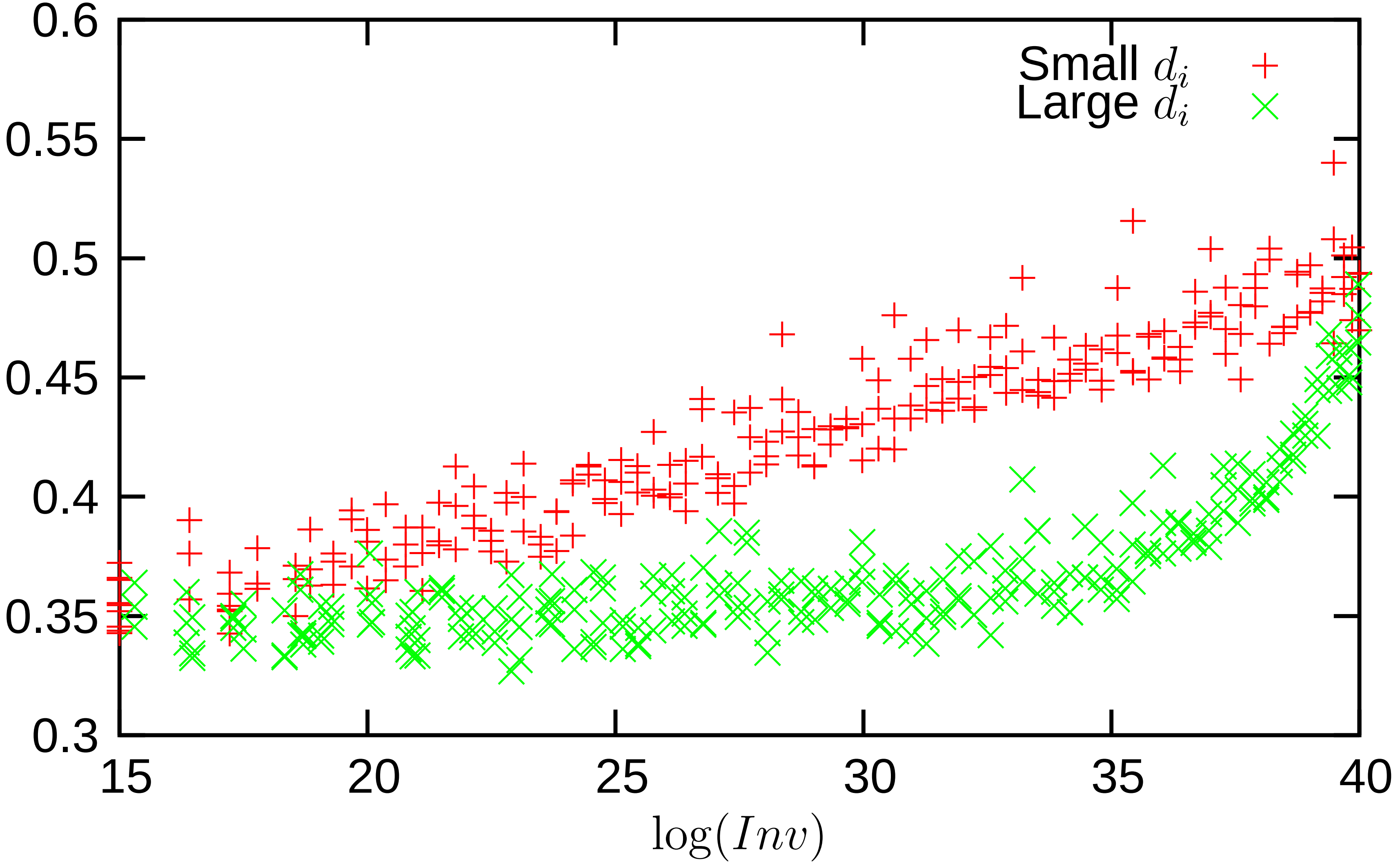
Quicksort (comparisons)

— Quicksort is not adaptive



Quicksort (running time)

— *Is Quicksort adaptive ?*



Results

Quicksort

- The number of comparisons is independent of the presortedness
- The number of swaps can be significantly smaller for nearly sorted inputs. We prove $O(n(1 + \log(1 + \frac{Inv}{n})))$.
- The number of branch mispredictions is given by the number of element swaps
- The running time is affected by more than a factor of two

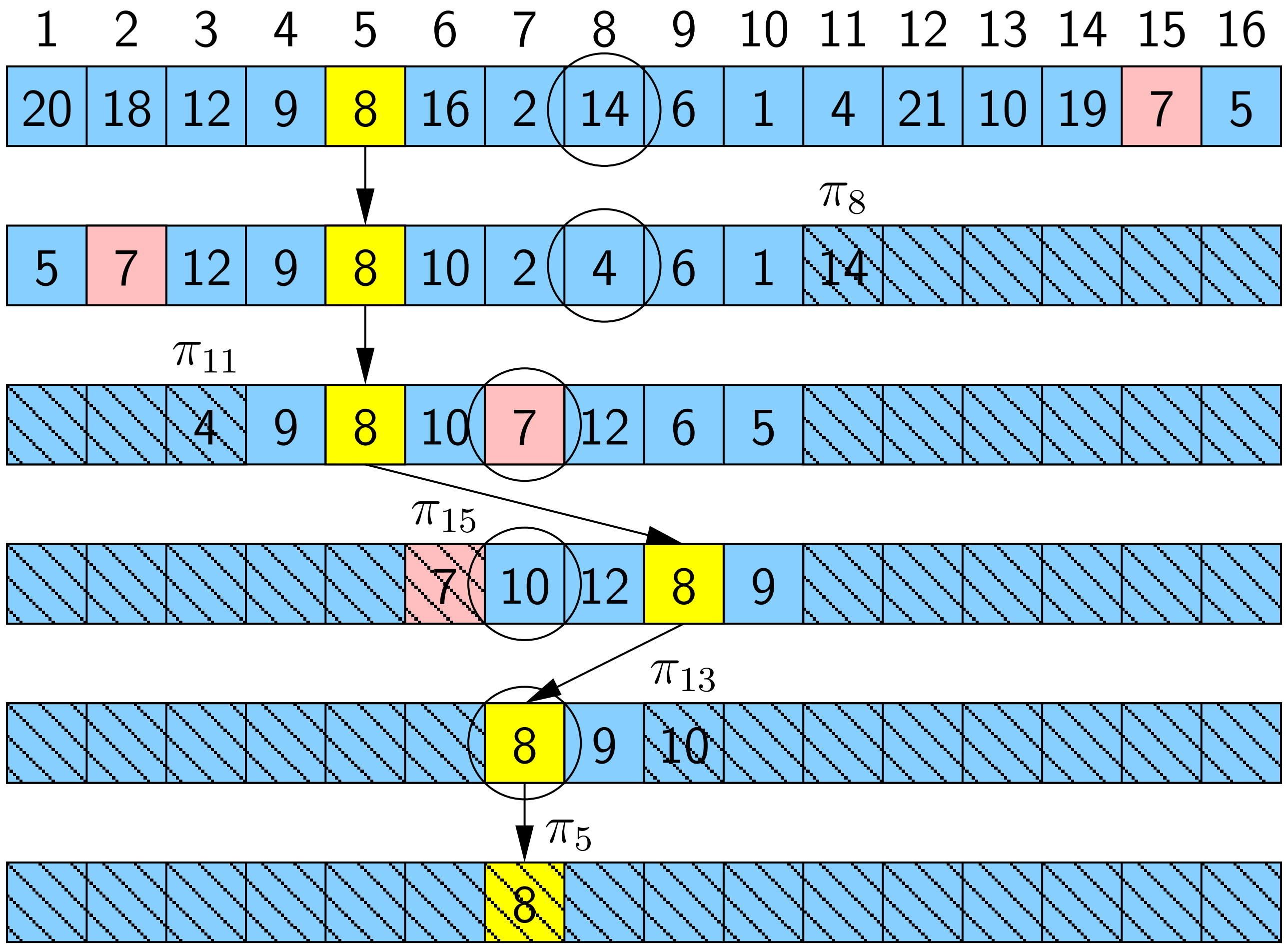
Binary Mergesort and Heapsort

- Empirical results are given

Quicksort C source

```
#define Item int
#define random(l,r) (l+rand() % (r-l+1))
#define swap(A, B) { Item t = A; A = B; B = t; }
void quicksort(Item a[], int l, int r)
   { int i;
     if (r <= l) return;
     i = partition(a, l, r);
     quicksort(a, l, i-1);
     quicksort(a, i+1, r);
   }
int partition(Item a[], int l, int r)
  { int i = l-1, j = r+1, p = random(l,r);
    Item v = a[p];
    for (;;) {
        while (++i < j && a[i] <= v);
        while (--j > i && v <= a[j]);
        if (j <= i) break;
        swap(a[i], a[j]);
      }
    if (p < i) i--;
    swap(a[i], a[p]);
    return i;
  }
```

Pivots vs Swaps



The first pivot causing $x_5 = 8$ to be swapped is $x_{15} = 7$

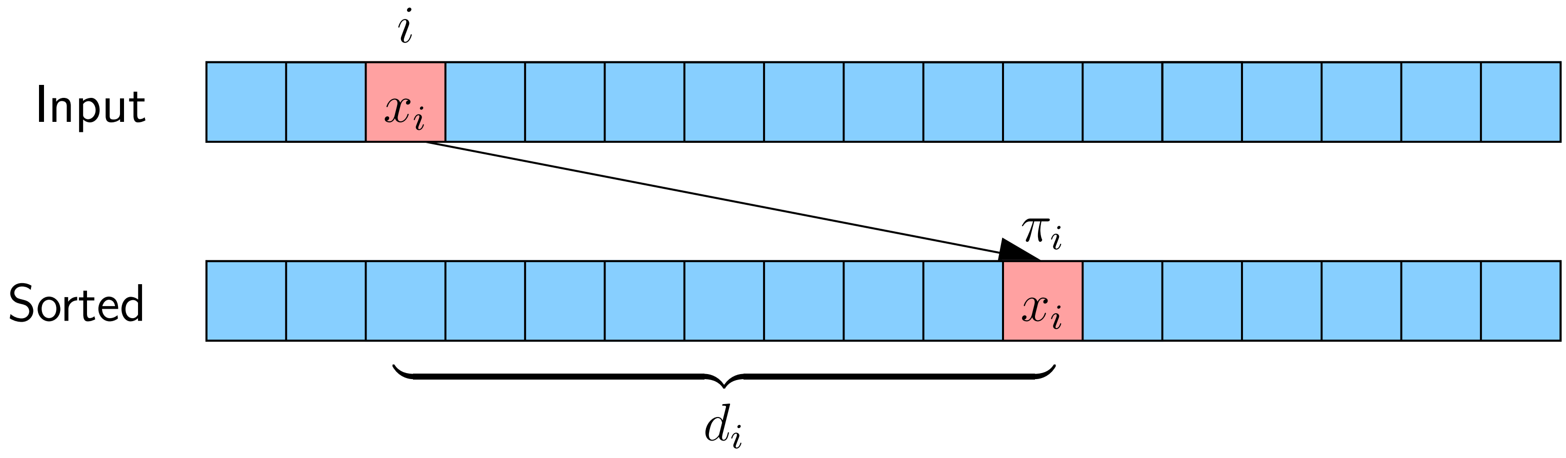
($\pi_5 = 7$, $\pi_{15} = 6$, and $5 \leq \pi_{15} < \pi_5$)

Main Theorem (I)

Theorem

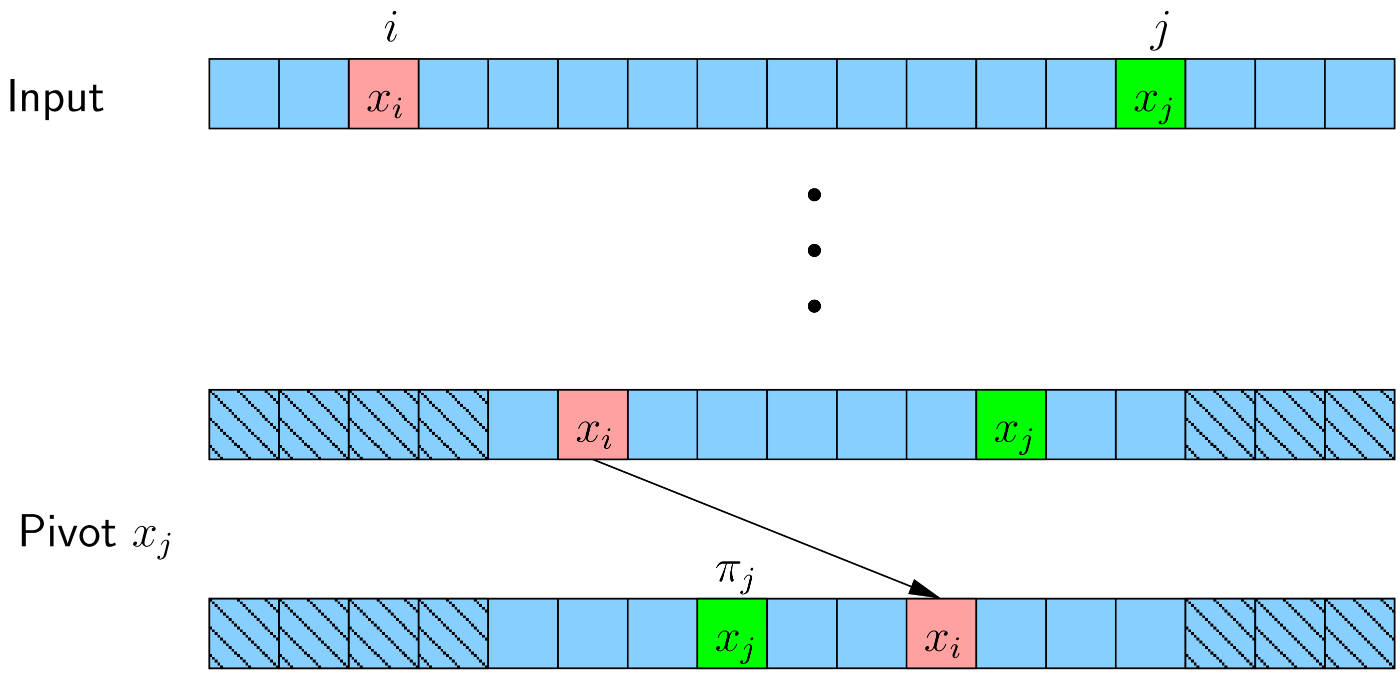
Quicksort performs expected $\leq n + n \ln \left(\frac{2Inv}{n} + 1 \right)$ swaps.

- $(x_1, \ldots, x_n)$ – input sequence of distinct elements
- π_i – rank of x_i in the sorted sequence
- $d_i = |\pi_i - i|$



Main Theorem (II)

Definition $X_{ij} = 1$ if when x_j becomes a pivot then x_i is swapped

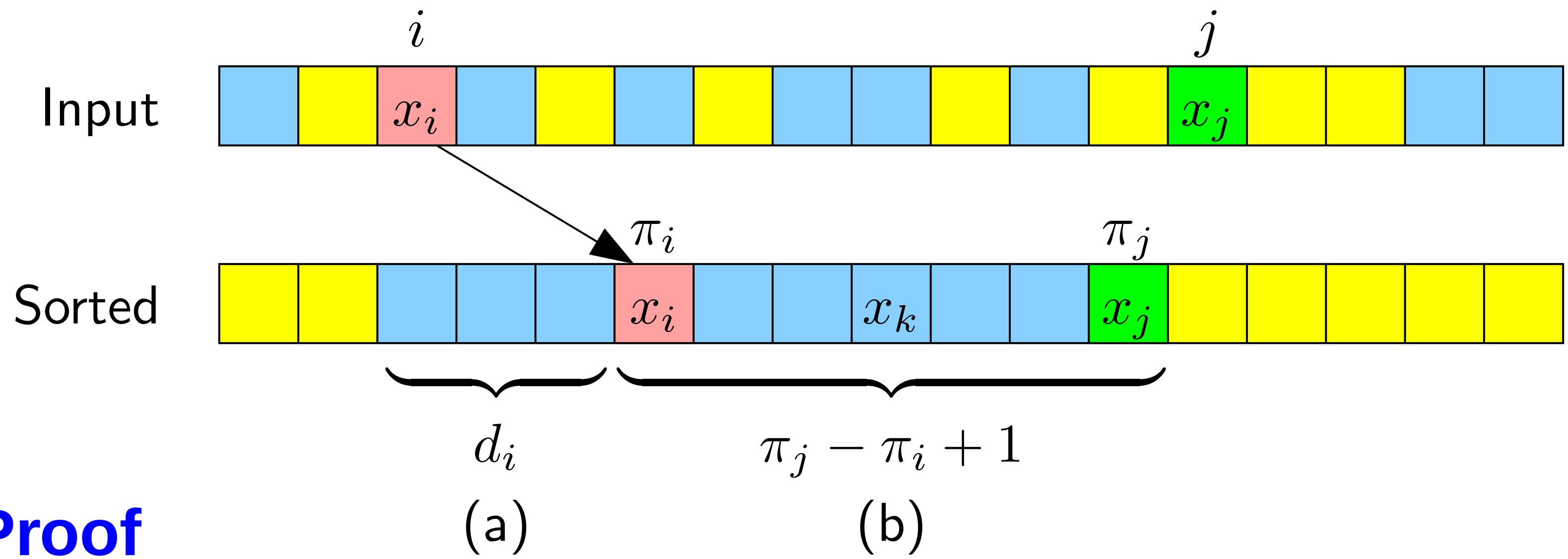


$$X_{ij} = 1$$

Main Theorem (III)

Lemma

$$\Pr[X_{ij}=1] \le \begin{cases} 0 & \text{if } \pi_j < i \le \pi_i \text{ or } \pi_i \le i < \pi_j \\ \frac{1}{|\pi_i-\pi_j|+1} & \text{if } i \le \pi_j < \pi_i \text{ or } \pi_i < \pi_j \le i \\ \frac{1}{|\pi_i-\pi_j|+1} - \frac{1}{|\pi_i-\pi_j|+1+d_i} & \text{otherwise} \end{cases}$$

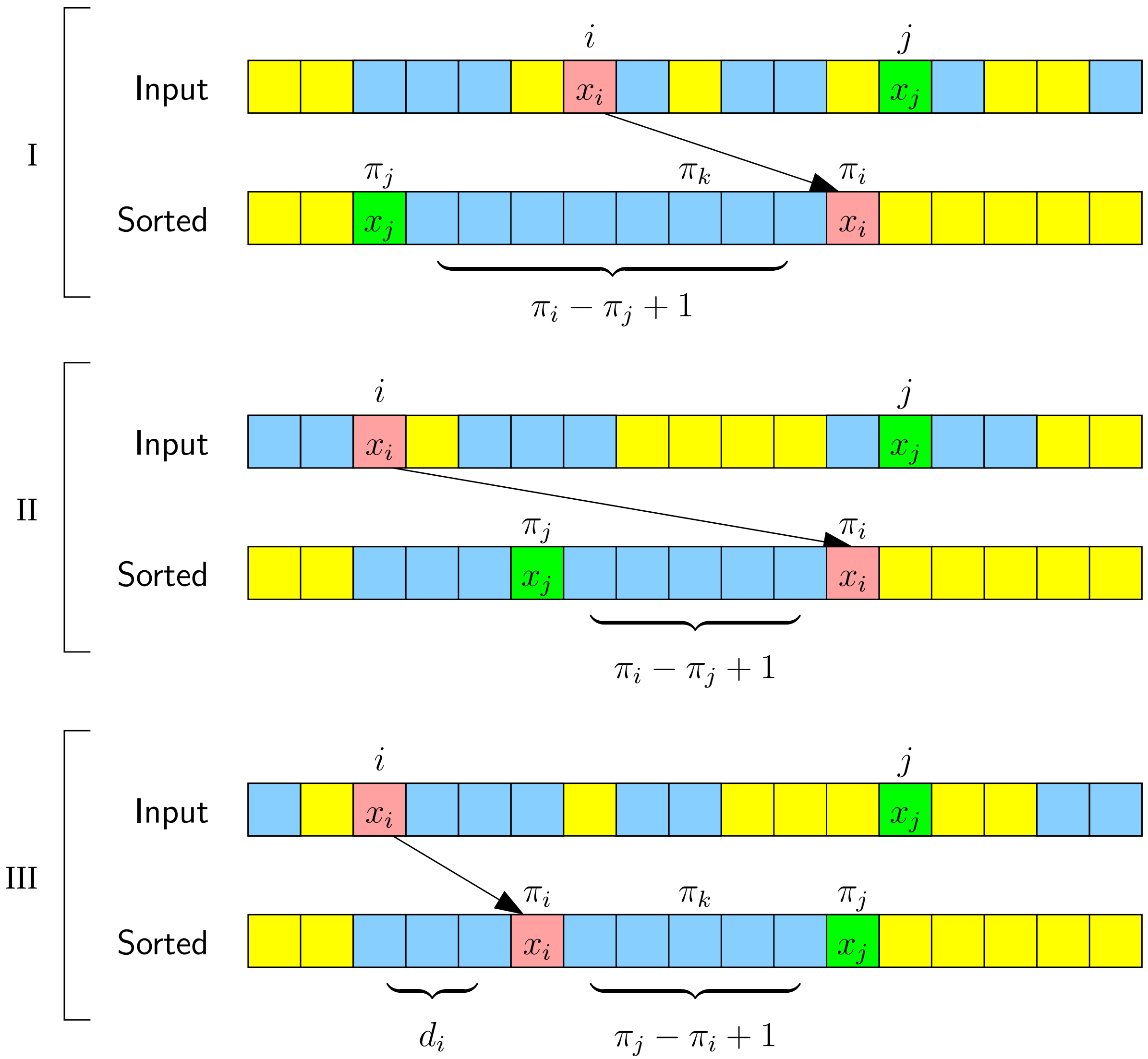


Proof

(a) Pivots forcing x_i to be swapped

(b) Pivots separating x_i and x_j

Main Theorem (IV)



$$Pr[X_{ij} = 1] = 0$$

$$Pr[X_{ij} = 1] \leq \frac{1}{|\pi_i - \pi_j| + 1}$$

$$Pr[X_{ij} = 1] \leq \frac{1}{|\pi_i - \pi_j| + 1} - \frac{1}{|\pi_i - \pi_j| + 1 + d_i}$$

Main Theorem (V)

Theorem

Quicksort performs expected $\leq n + n \ln\left(\frac{2Inv}{n} + 1\right)$ swaps.

Proof

$$\mathrm{E}\left[\sum_{j=1}^{n}\left(1 + \frac{1}{2}\sum_{i=1, i\neq j}^{n} X_{ij}\right)\right] = n + \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1, i\neq j}^{n} \Pr(X_{ij} = 1)$$

$$\leq n + \frac{1}{2}\sum_{i=1}^{n}\left(\sum_{k=1}^{d_i}\frac{1}{k+1} + \sum_{k=1}^{n}\left(\frac{1}{k+1} - \frac{1}{k+1+d_i}\right)\right)$$

$$\leq \sum_{i=1}^{n}\sum_{k=1}^{d_i+1}\frac{1}{k} \leq n + n\ln\left(\frac{2Inv}{n} + 1\right)$$

using $\sum_{i=1}^{n} d_i \leq 2Inv$

Experimental Setup

- Two types of input

  (1) x_i uniformly at random in $[i-d..i+d]$ for increasing d, i.e. small d_i

  (2) $x_i = i$ except for some random i where x_i is randomly in $[0..n-1]$, i.e. large d_i

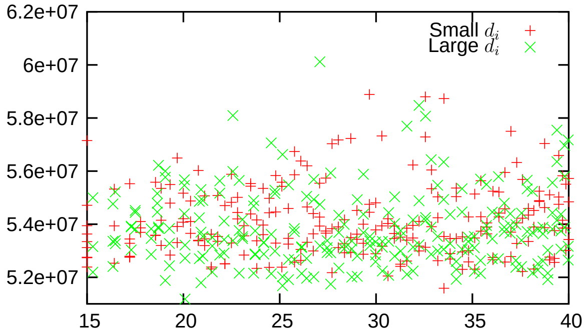
- Compare #comparisons, #swaps, #branch mispredictions, #L2 data cache misses and the running time against $\log(Inv)$

- $n = 2 \times 10^6$

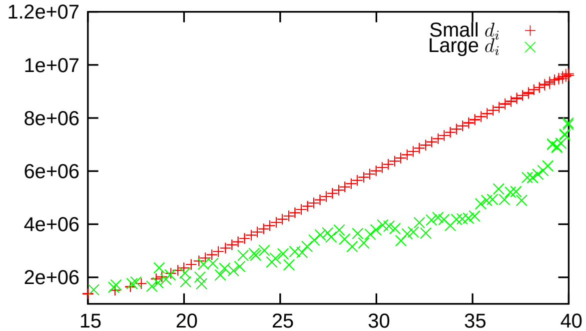
- AMD Athlon XP 2400+ 2.0 GHz, Redhat 9, Linux 2.4.20, gcc 3.3.2 using optimization -O3, PAPI 3.0

Experimental results (Quicksort)

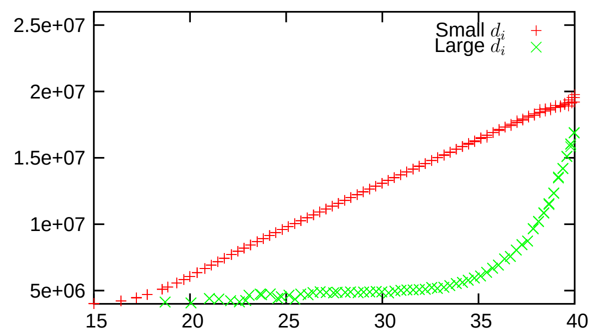
Comparisons
(10% difference)



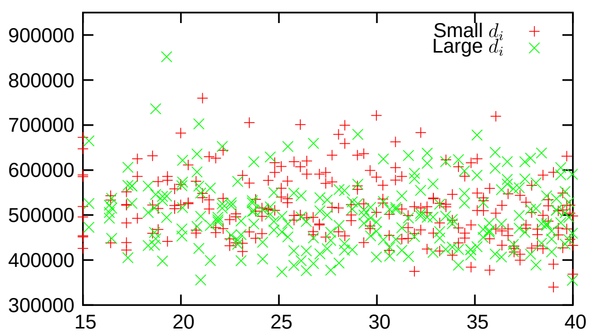
Swaps
(500% difference)



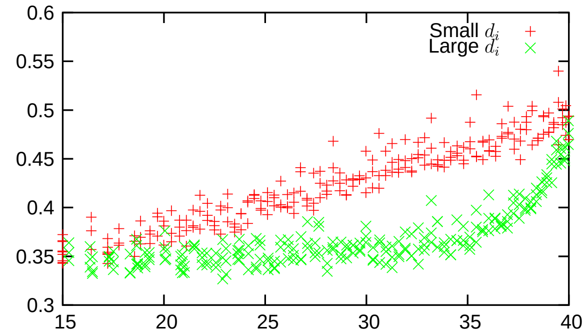
Mispredictions
(400%)



Cache misses
(60% difference)

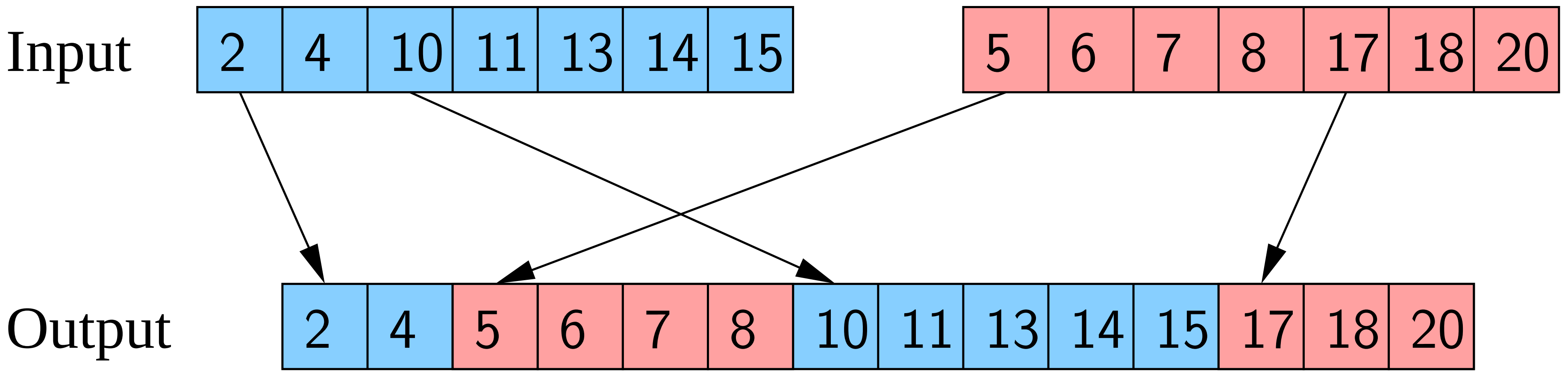


Running time
(50% difference)



Binary Mergesort

Alternations - the number of changes between the two input sequences in the result of a binary merging

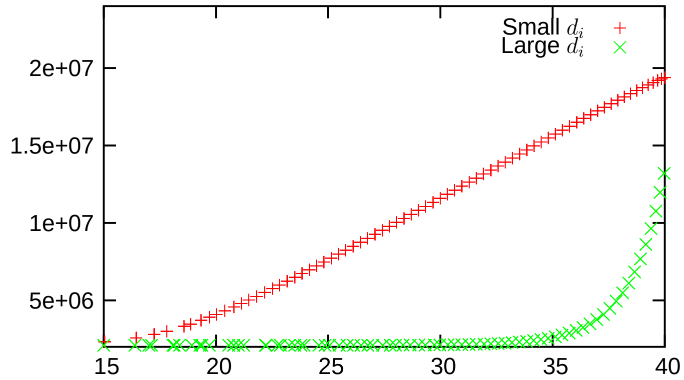


By result of Moffat et al.:

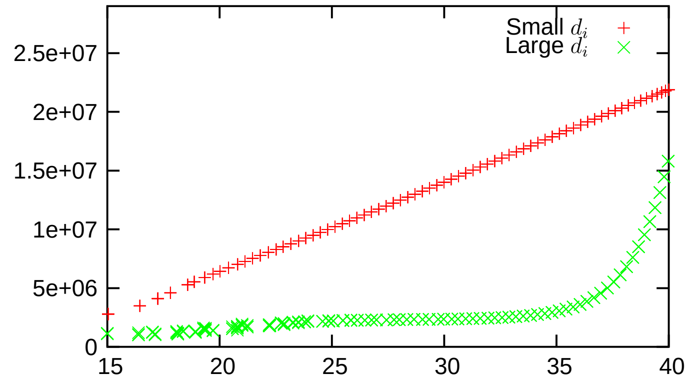
The number of alternations for Mergesort is $O(n \log \frac{Inv}{n})$

Experimental results (Mergesort)

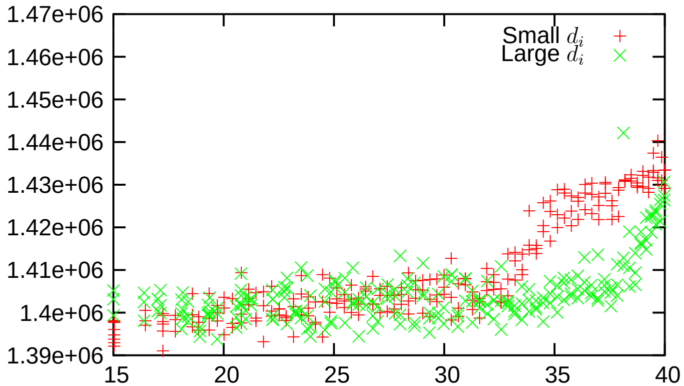
Alternations
(900% difference)



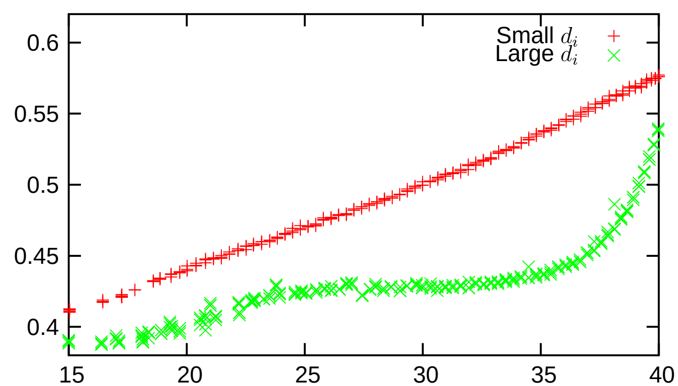
Mispredictions
(900% difference)



Cache misses
(5% difference)

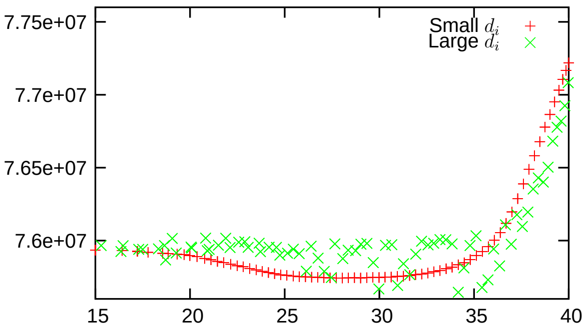


Running time
(35% difference)

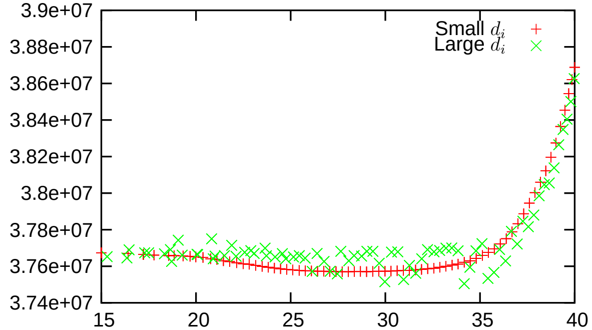


Experimental results (Heapsort)

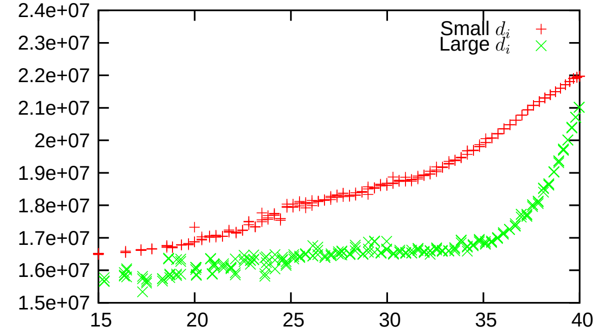
Comparisons
(5% difference)



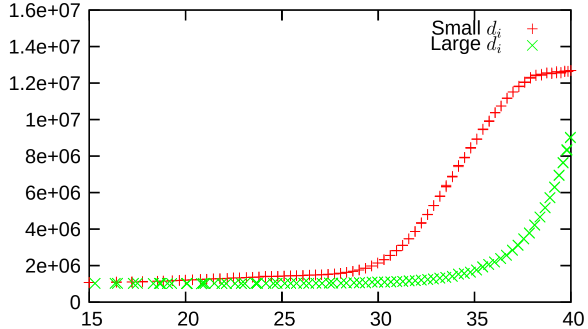
Swaps
(18% difference)



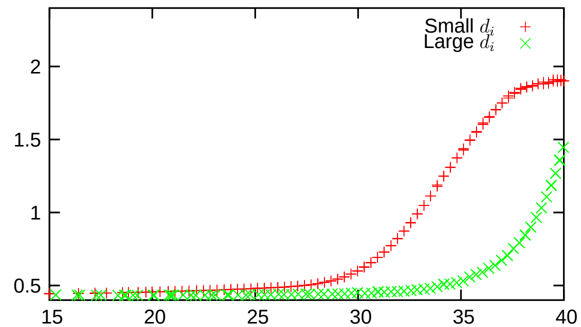
Mispredictions
(difference 30%)



Cache misses
(1000% difference)



Running time
(400% difference)



Conclusions

- Randomized Quicksort performs expected $O(n(1 + \log(1 + Inv/n)))$ swaps
- The number of branch mispredictions is given by the number of swaps
- The number of swaps performed can affect the running time of Quicksort by more than a factor of two
- Experimental results confirm the theoretical results for Quicksort
- Empirical results are given for Heapsort and Binary Mergesort