On the Adaptiveness of Quicksort

Gabriel Moruz
BRICS
University of Aarhus

Joint work with Gerth Stølting Brodal and Rolf Fagerberg

ALENEX05, Vancouver, British Columbia, Canada, January 22, 2005
Quicksort

- Introduced by Hoare in 1961
- Simple, randomized sorting algorithm
- Expected number of comparisons $\sim 1.4n \log_2 n$ [Hoare’62]
- Average number of swaps for a random input is $1/6$ the expected number of comparisons [Hoare’62]
- Elements are compared and swapped within the input array “implicit sorting algorithm” (except for the runtime stack)
Adaptiveness

- Adaptive sorting - the running time depends both on the input size and the presortedness in the input

- A common measure of presortedness:

\[ \text{Inv}(x_1 \ldots x_n) = |\{(i, j) \mid i < j \land x_i > x_j\}| \]

\[ \text{Inv}(1, 2, 3, 4) = 0, \text{Inv}(4, 3, 2, 1) = 6, \text{Inv}(2, 1, 4, 3) = 2 \]

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

— Quicksort is not adaptive
Quicksort (running time)

—is Quicksort adaptive?

Gabriel Moruz: On the Adaptiveness of Quicksort
Results

Quicksort

- The number of comparisons is independent on the presortedness
- The number of swaps can be significantly smaller for nearly sorted inputs (we prove $O(n(1 + \log(1 + \frac{\text{Inv}}{n})))$)
- 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
```c
#include <stdio.h>
#include <stdlib.h>

#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;
}
```

Gabriel Moruz: On the Adaptiveness of Quicksort
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 \( \leq n + n \ln \left( \frac{2\text{Inv}}{n} + 1 \right) \) swaps.

- \((x_1, \ldots, x_n)\) – input sequence of distinct elements
- \(\pi_i\) – 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**

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

**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 \( \leq n + n \ln \left( \frac{2 \text{Inv}}{n} + 1 \right) \) swaps.

Proof

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

\[
\leq \sum_{i=1}^{n} \sum_{k=1}^{d_i+1} \frac{1}{k}
\]

\[
\leq n + n \ln \left( \frac{2 \text{Inv}}{n} + 1 \right)
\]

using \( \sum_{i=1}^{n} d_i \leq 2 \text{Inv} \)
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

<table>
<thead>
<tr>
<th>Input</th>
<th>2</th>
<th>4</th>
<th>10</th>
<th>11</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>17</th>
<th>18</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output</td>
<td>2</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>10</td>
<td>11</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>17</td>
<td>18</td>
<td>20</td>
</tr>
</tbody>
</table>

**Result:**
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