On the Adaptiveness of Quicksort

Gabriel Moruz

BRICS
University of Aarhus

Joint work with Gerth Stølting Brodal and Rolf Fagerberg

Schloss Dagstuhl, Germany, July 22, 2004
ALGORITHM 63
PARTITION
C. A. R. HOARE
Elliott Brothers Ltd., Borehamwood, Hertfordshire, Eng.

procedure partition (A,M,N,I,J); value M,N;
array A; integer M,N,I,J;

comment I and J are output variables, and A is the array (with subscript bounds M:N) which is operated upon by this procedure.
Partition takes the value X of a random element of the array A, and rearranges the values of the elements of the array in such a way that there exist integers I and J with the following properties:

M ≤ J < I ≤ N provided M < N
A[R] ≤ X for M ≤ R ≤ J
A[R] = X for J < R < I
A[R] ≥ X for I ≤ R ≤ N

The procedure uses an integer procedure random (M,N) which chooses equiprobably a random integer F between M and N, and also a procedure exchange, which exchanges the values of its two parameters;

begin real X; integer F;
F := random (M,N); X := A[F];
I := M; J := N;
up: for I := 1 step 1 until N do
    if X < A[I] then go to down;
    I := N;
down: for J := J step -1 until M do
    if A[J] < X then go to change;
    J := M;
change: if I < J then begin exchange (A[I], A[J]);
       I := I + 1; J := J - 1;
       go to up end
else if I < F then begin exchange (A[I], A[F]);
     I := I + 1 end
else if F < J then begin exchange (A[F], A[J]);
     J := J - 1 end;
end; partition

ALGORITHM 64
QUICKSORT
C. A. R. HOARE
Elliott Brothers Ltd., Borehamwood, Hertfordshire, Eng.

procedure quicksort (A,M,N); value M,N;
array A; integer M,N;

comment Quicksort is a very fast and convenient method of sorting an array in the random-access store of a computer. The entire contents of the store may be sorted, since no extra space is required. The average number of comparisons made is 2(M−N) ln (N−M), and the average number of exchanges is one sixth this amount. Suitable refinements of this method will be desirable for its implementation on any actual computer;

begin integer I,J;
    if M < N then begin partition (A,M,N,I,J);
        quicksort (A,M,I);
        quicksort (A,I,N)
    end
end quicksort

Gabriel Moruz: On the Adaptiveness of Quicksort
Quicksort

- Introduced by Hoare in 1961
- Simple, randomized sorting algorithm
- Elements are compared and swapped within the input array “implicit sorting algorithm” (except for the runtime stack)
- 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]

This talk

- Characterize the expected number of swaps performed by Quicksort by the amount of disorder (inversions) in the input

$$O(n(1 + \log(1 + Inv/n)))$$
Adaptiveness

- What if the input is nearly sorted?
- Adaptive sorting - the running time depends both on the input size and the presortedness in the input
- A common measure of presortedness:
  \[ \text{Inv}(x_1 \ldots x_n) = |\{(i, j) \mid i < j \land x_i > x_j\}| \]
- An optimal sorting algorithm with respect to Inv performs \( \Theta(n(1 + \log(1 + \frac{\text{Inv}}{n]))) \) comparisons

Quicksort

- The expected number of comparisons performed is independent of the order of the input (expected \( O(n \log n) \))
- The number of swaps can be significantly smaller for nearly sorted inputs (we prove \( O(n(1 + \log(1 + \frac{\text{Inv}}{n}))) \))
```c
#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 \( \leq n + n \ln \left( \frac{4\text{Inv}}{n} + 1 \right) \) swaps.

- \((x_1, \ldots, x_n)\) — input sequence of distinct elements
- \(\pi_i\) — rank of \(x_i\) in the sorted sequence
- \(d_i = |\pi_i - i|\)
- \(X_{ij} = 1\) if when \(x_j\) becomes a pivot then \(x_i\) and \(x_j\) are in the same set and \(x_i\) is swapped
Main Theorem (II)

Lemma

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

Proof

Input

\[ x_i \]

\[ x_j \]

\[ \pi_i \]

\[ \pi_j \]

Sorted

\[ x_i \]

\[ x_k \]

\[ x_j \]

\[ d_i \]

\[ \pi_j - \pi_i + 1 \]

(a) Pivots forcing \( x_i \) to be swapped

(b) Pivots separating \( x_i \) and \( x_j \)
Main Theorem (III)

Theorem

Quicksort performs expected $\leq n + n \ln \left( \frac{4 \text{Inv}}{n} + 1 \right)$ swaps.

Proof

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

$$
\leq \sum_{i=1}^{n} \sum_{k=1}^{2d_i + 1} \frac{1}{k}
$$

$$
\leq n + n \ln \left( \frac{4 \text{Inv}}{n} + 1 \right)
$$

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, and the running times against $\log \frac{Inv}{n}$

- $n = 2 \times 10^6$

- Intel P4 3.0 GHz, Redhat 9, Linux 2.4.20, gcc 3.3.2 using optimization -O3.
Number of Comparisons

(1) Small $d_i$
(2) Large $d_i$
Number of Swaps

(1) Small $d_i$  
(2) Large $d_i$
Running Time

(1) Small $d_i$  
(2) Large $d_i$

![Graph showing running time for small and large $d_i$ values](graph.png)
Summary of Experimental results

Comparisons

Swaps

Running time

Gabriel Moruz: On the Adaptiveness of Quicksort
Conclusions

- Quicksort performs expected $O(n(1 + \log(1 + Inv/n)))$ swaps
- The number of branch mispredictions is given by the number of swaps
- $\#\text{swaps}/B \leq \#\text{cache faults}(\text{write})/2 \leq \#\text{swaps}$
- The number of swaps performed can affect the running time of Quicksort by up to a factor of two
- Empirical results confirm the theoretical results