# W4231: Analysis of Algorithms

9/9/1999

- Master Theorem
- Max-Min



## Divide and Conquer

**Divide:** reduce the instance of the problem to be solved to instances of smaller size.

**Conquer:** use recursion to solve the smaller instances.

**Combine:** use solutions for the smaller instances to find a solution for the original instance.



## Mergesort

**Problem:** to sort a given array $a_1, \ldots, a_n$.

**Divide:** divide the sequence into the two subsequences $a_1, \ldots, a_{n/2}$ and $a_{n/2+1}, \ldots, a_n$.

**Conquer:** sort the two sequences.

**Combine:** merge the sorted subsequences.



## Quicksort

**Problem:** Ditto.

**Divide:** select an element $a$ from the sequence. Partition the sequence into the set of elements $\leq a$ and the set of elements $> a$.

**Conquer:** sort the two sequences.

**Combine:** paste the sorted subsequences.



## Binary Search

Look for $a$ into $A[1, \ldots, n]$.

Special case of looking for $a$ into $A[i, \ldots, j]$.

If $i > j$ then `not found`

Else if $a = A\left[\left\lfloor \frac{i+j}{2} \right\rfloor\right]$ then `found!`.

Else if $a > A\left[\left\lfloor \frac{i+j}{2} \right\rfloor\right]$ then look for $a$ in $A\left[\left\lfloor \frac{i+j}{2} \right\rfloor, \ldots, j\right]$.

Else look for $a$ in $A\left[i, \ldots, \left\lfloor \frac{i+j}{2} \right\rfloor\right]$.



## Recurrence relation

In Mergesort, if $T(n)$ is the number of operations, then

$$T(1) = 1 \text{ and } T(n) = 2T(n/2) + cn$$

where $c$ is a constant.

Then it must be $T(n) = \Theta(n \log n)$.

## Master Theorem

Let $a, b, c \geq 1$ constants. Let $\beta = \log_b a$. Let $f(\cdot)$ be a positive function, and $T(\cdot)$ be a function over the integers defined as

$$T(1) = c \text{ and } T(n) = aT(n/b) + f(n)$$

then

1. If $f(n) = O(n^{\beta'})$ with $\beta' < \beta$, then $T(n) = \Theta(n^\beta)$.
2. If $f(n) = \Theta(n^\beta)$ then $T(n) = \Theta(n^\beta \log n)$.



3. If $f(n) = \Omega(n^{\beta'})$ with $\beta' > \beta$ and $af(n/b) < cf(n)$ for some $c < 1$, then $T(n) = \Theta(f(n))$.



## Examples

Always assume $T(1) = O(1)$.

1. If $T(n) = 7T(n/2) + O(n^2)$ then $T(n) = \Theta(n^{\log_2 7}) = \Theta(n^{2.807\dots})$ (matrix multiplication).
2. If $T(n) = 2T(n/2) + O(n)$ then $T(n) = \Theta(n \log n)$ (mergesort).
3. If $T(n) = 2T(n/2) + n^2$ then $T(n) = O(n^2)$ (made-up example).



## Max and Min

Given a list of integers $a_1, \dots, a_n$, we want to find the minimum *and* the maximum.

We can do that in linear time, and this is optimal.

Let us get a better grip on the precise *number of comparisons*.



## How many comparisons?

Simple algorithm:

(1) Find the minimum with $n - 1$ comparisons.

```
curr_min = a[1];
for (i=2; i<= n; i++)
  if (a[i] < curr_min)  curr_min = a[i];
return curr_min;
```

(2) Find the maximum with other $n - 1$ comparisons.

Total: $2n - 2$ comparisons.



## Divide and Conquer

Let $m_1$ and $M_1$ be the minimum and the maximum of the subsequence $a_1, \dots, a_{n/2}$.

Let $m_2$ and $M_2$ be the minimum and the maximum of the subsequence $a_{n/2+1}, \dots, a_n$.

Let $m = \min(m_1, m_2)$ and $M = \max(M_1, M_2)$.

Then $m$ is the minimum of $a_1, \dots, a_n$, and $M$ is the maximum of $a_1, \dots, a_n$.

## Analysis

$C(n) = 2C(n/2) + 2$, $C(2) = 1$, $C(1) = 0$.

Solving by substitution. By the Master Theorem we know $C(n) = \Theta(n)$, therefore we try and write $C(n) = an + b$. We derive

$$an + b = 2(an/2 + b) + 2$$
$$2a + b = 1$$

Which solves to $a = 1.5$ and $b = -2$. And $C(n) = \lceil 3n/2 \rceil - 2$ solves the recurrence for every $n$.



## Optimality

$3n/2 - 2$ comparisons is the best possible.

The proof is somewhat complicated (and we do not do it in full).

It is simpler to first prove that $n - 1$ comparisons are necessary to find the maximum.



## Lower Bounds for the Maximum

It is simpler to first prove weaker lower bound.

A straightforward decision-tree argument shows that $\log n$ comparisons are necessary. (Why?)

It is also "easy" to see that $n/2$ comparisons are necessary.



## The Precise Lower Bound for the Maximum

Fix an algorithm for finding the maximum of $n$ integers. We show that the algorithm makes $n - 1$ comparisons (or makes a mistake).

Observe the behavior of the algorithm on input a certain seqeunce (say, a sorted sequence). While the algorithm runs, maintain a list of possible positions that could be the maximum compatibly with the comparisons done so far.

The list contains initially $n$ elements. When the algorithm finishes the list contains $1$ element. Each comparison can cancel at most one element from the list.



## Lower Bound for Max-Min

Consider any algorithm. Fix an even $n$. We will construct an instance where the algorithm makes $3n/2 - 2$ comparisons.

During the execution of the algorithm, maintain the list of elements that could be the maximum and the list of elements that could be the minimum.

Initially $2n$ total elements, at the end $2$.

Comparisons between two elements in both lists: two cancellations (at most $n/2$ times).

Other comparisons: we can force 1 or 0 cancellations.



## Without recursion (and without fun)

```
if (a[1]<a[2])
  {m=a[1]; M=a[2];}
else
 {m=a[2]; M=a[1];}
for (i=3; i<=n-1; i+=2){
  if (a[i]<a[i+1])
    {tm=a[i]; tM=a[i+1];}
  else
    {tm=a[i+1]; tM=a[i];}
  if (tM > M) M=tM;
  if (tm < m) m=tm;
}
```