CS512 LECTURE NOTES - LECTURE 7

0.1 Analysis of Mergesort

Time complexity

In the case of Mergesort, the analysis is straight forward for best, worst and average cases, since the running time of the Merge algorithm is always in $\Theta(n)$. Therefore, the algorithm requires two recursive calls to itself with lists of half its the original size, as well as a call to Merge, so we get the following recurrence relation:

$$T(n) = 2T\left(\frac{n}{2}\right) + \Theta(n)$$

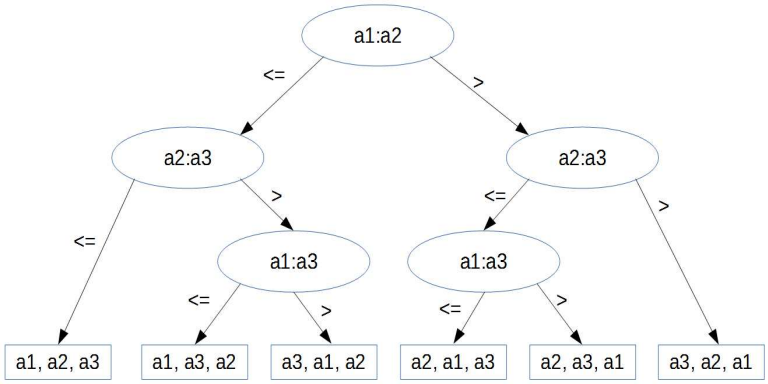
This recurrence relation can be easily solved using the Master Theorem:

$$T(n) \in \Theta(n \lg n)$$

1 Lower bound on Sorting

1.1 Comparison Decision Trees

When sorting BY COMPARISONS we need to make as many comparisons as needed to be able to determine the sorted permutation of the input list of numbers. This process can be illustrated using a "Comparison Decision Tree" as shown in the figure.



In the example an algorithm must determine the sorted permutation of a list of three numbers. Notice that the internal nodes of the decision tree represent key comparisons and the leafs of the tree represent permutations.

The total number of permutations of a list of n elements is $n!$, we have that the height h of the tree must satisfy:

$$h \geq \lg n!$$

1.2 The lower bound

From the Stirling approximation we know that

$$n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$$

Taking logarithm on both sides

$$\lg n! \approx \lg\sqrt{2\pi} + \frac{1}{2}\lg n + n\lg n - n\lg e$$

So we have that

$$\lg n! \in \Theta(n\lg n)$$

Therefore

$$h \in \Omega(n\lg n)$$

Since ANY sorting algorithm that uses comparisons to sort MUST perform at least as many comparisons as the height (h) of the comparison decision tree, we can conclude:

$n\lg n$ is a lower bound for comparison-based sorting.

2 Selection

The selection problem consists of finding the i-th smallest element of an array.

2.1 Min and Max

Finding the smallest element of an array (min) requires $\Theta(n)$ time, as does finding the largest element of an array (max).

The number of comparisons required to find the maximum is $n-1$, and the number of comparisons required to find the minimum is also $n-1$. However, to find the maximum AND the minimum can be done in $\frac{3}{2}n$: Divide the list into sublists of size two. For each one of the $n/2$ sublists, find the min and the max. Then find the max of all of the maximums, and the min of all the minimums. This requires in total $n/2$ comparisons to find the max and min of each sublist of size 2, and then $n/2$ comparisons to find the max of all the maximums and $n/2$ comparisons to find the min of all the minimums, so the algorithm requires $\frac{3}{2}n$ comparisons.

This shows that a simple idea can be extremely useful in making algorithms run faster.

2.2 Simple Selection

Finding the k-th smallest element can be done easily with the following algorithm:
Algorithm Selection(a, k)

1. Sort A
2. return A[k]

This algorithm runs in $O(n \lg n)$. But why does finding min, and finding max, which are special cases of the selection problem when k=1 and when k=n respectively, require only linear time?

Is it possible to design a better version o Select that has a worst case complexity of $\Theta(n)$?

2.3 Randomized Divide and Conquer algorithm for the Selection problem

Algorithm Select(a,$first$,$last$,k)
q =ranomizedPartition(a,$first$,$last$)

1. if $k = q$ then return $a[q]$
2. if $k < q$ then return Select(a,$first$,$q-1$,k)
3. if $k > q$ then return Select(a,$q+1$,$last$,k)

The idea is to use partition to divide the array in two parts: from $first$ to $q-1$ and from $q+1$ to $last$. If the k-th smallest element is in the first part ($k < q$) we recursively search on the first part, if $k > q$ then we search on the second part.

The problem with this approach is that partition might not divide the array evenly.

Analysis

We consider a good split, where 1/4 of the elements end up on one side and 3/4 of the elements end up on the other side.

Notice that the probability of obtaining a good split is 0.5. Therefore, the expected number of times that we need to invoke partition to get a good split is equivalent to the expected number of times that we need to flip a coin to get a head:

$$E = 1 + \frac{1}{2}E$$

Where E is the required expected value, the expected value is 1 with probability 1, and E (after the first flip we did not get head) with probability 1/2.

Solving for E we get that E=2.

So on average we need to invoke partition twice to get a good split!

The recurrence relation that we would get for the randomized version of **Select** is:

$$T(n) = T(3n/4) + \Theta(n)$$

Because after two partition on average we have a good split, and the time complexity of the randomized partition algorithm is $\Theta(n)$.

Solving this recurrence relation using Master's Theorem we get that

$$T(n) \in \Theta(n)$$

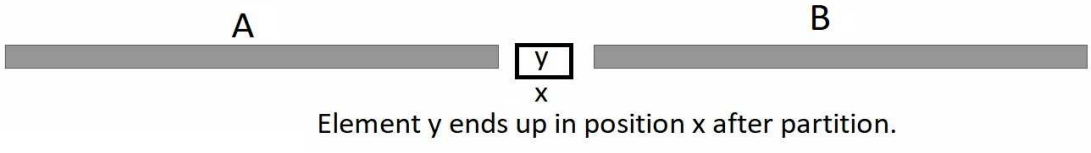
Therefore **Select** runs in linear time on average.

2.4 Selection in worst case linear time

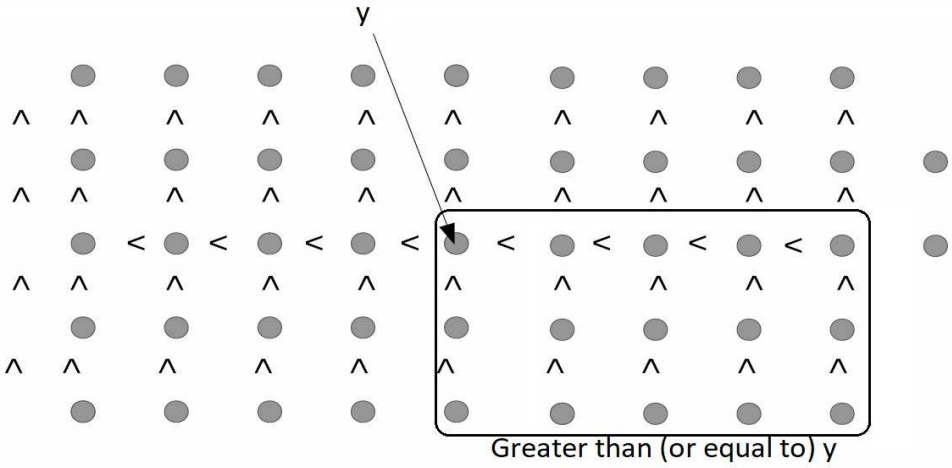
Algorithm Selection(a,$first$,$last$,k)

1. Divide a in groups of 5
2. Find the median of each group (using insertion sort)
3. Recursively compute y, the median of the medians
4. Partition the array using y as pivot
5. Let q the position of the pivot after partitioning
   (a) if $k = q$ then return $a[q]$
   (b) if $k < q$ then return Select(a,$first$,$q-1$,k)
   (c) if $k > q$ then return Select(a,$q+1$,$last$,k)

Partition using y as pivot:



To help us analyze the algorithm we can use the following figure:



Notice that the number of elements in the rectangle shown in the picture must be greater than

$$3\left(\frac{1}{2}\left\lceil\frac{n}{5}\right\rceil\right) = \frac{3}{10}n$$

Therefore, the largest side of A and B must be less than or equal to:

$$\max\{|A|, |B|\} \leq \frac{7}{10}n$$

We now provide the running time of each one of the steps of the algorithm:

1. 0 No actual work is necessary
2. $\Theta(n)$ (the number of comparisons to sort an array with 5 elements using insertion sort is 10)
3. $T(n/5)$
4. $\Theta(n)$
5. $T(\frac{7}{10}n)$

So we have the following recurrence relation:

$$T(n) \leq T\left(\frac{n}{5}\right) + T\left(\frac{7}{10}n\right) + an$$

Where $a > 0$ is a known constant.