CPSC 320: Divide and Conquer Algorithms[*]

Invented by Tony Hoare in the late 1950's, the Quicksort algorithm was a breakthrough in sorting methods. Variants of Hoare's original algorithm are still a sorting method of choice today. In this worksheet, you will gain experience with the divide and conquer algorithmic design approach, as well as the analysis of recurrence relations, that led Hoare to this breakthrough. You will apply this algorithm in the next worksheet to the problem of finding the median of a list of values.

1 The Master Theorem

For each of the following recurrence relations, determine whether or not the Master Theorem can be used. If it can, give the solution to the recurrence. If it can not, explain why not. In all cases, assume that $T(n) \in \Theta(1)$ when n is sufficiently small.

1. $T(n) = 5T(\sqrt{n}) + n$

2. $T(n) = T(n/2) + 1$

3. $T(n) = T(n/4) + n$

[*]Copyright Notice: UBC retains the rights to this document. You may not distribute this document without permission.

4. $T(n) = 3T(n/9) + \sqrt{n} \log_2 n$

5. $T(n) = \sqrt{n}T(n/3) + n^2$

6. $T(n) = 9T(n/3) + n \log n$

7. $T(n) = 2T(n/2) + n/\log n$

2 Quicksort Runtime Analysis

Here is a basic version of Quicksort. We will assume that the array $A[1..n]$ to be sorted has n distinct elements.

```
function QUICKSORT($A[1..n]$)        ▷ returns the sorted array $A$ of $n$ distinct numbers
    if $n > 1$ then
        Choose pivot element $p = A[1]$
        Let Lesser be an array of all elements from $A$ less than $p$
        Let Greater be an array of all elements from $A$ greater than $p$
        LesserSorted $\leftarrow$ QuickSort(Lesser)
        GreaterSorted $\leftarrow$ QuickSort(Greater)
        return the concatenation of LesserSorted, $[p]$, and GreaterSorted
    else
        return $A$
```

1. Suppose that QuickSort happens to always select the $\lceil \frac{n}{4} \rceil$-th smallest element as its pivot. Give a recurrence relation for the running time of QuickSort.

2. Using the recurrence, draw a recursion tree for QuickSort. Label each node by the number of elements in the array at that node's call (the root is labeled n) and the amount of time taken by that node but not its children. Also, label the total work (time) for each "level" of calls. Show the root at level 0 and the next two levels below the root, and also the node at the leftmost and rightmost branches of level i.

3. Find the following two quantities.

   (a) The number of levels in the tree down to the shallowest leaf. *Hint:* Is the shallowest leaf on the leftmost side of the tree, the rightmost side, or somewhere else? If you've already described the problem size of the leftmost and rightmost nodes at level i as a function of i, then set that equal to the problem size you expect at the leaves and solve for i.

   (b) The number of levels in the tree down to the deepest leaf.

4. Use the work from the previous parts to find asymptotic upper and lower bounds for the solution of your recurrence.

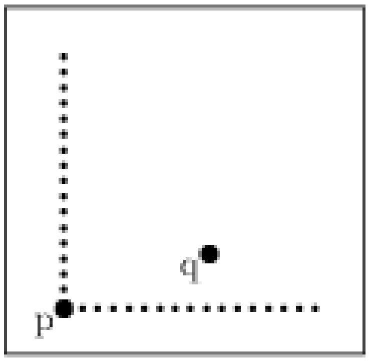
5. Now, we will relax our assumption that Quicksort always selects the $\lceil \frac{n}{4} \rceil$-th smallest element as its pivot. Instead, consider a weaker assumption that the rank of the pivot is always in the range between $\lceil \frac{n}{4} \rceil$ and $\lfloor \frac{3n}{4} \rfloor$ (the *rank* of an element is k if the element is the kth smallest in the array). What can you say about the running time of Quicksort in this case?

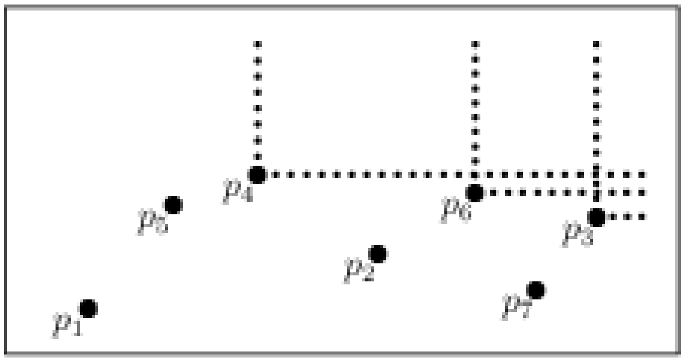
3 The Stock Market, dividends and risks

A firm of financial analysts has hired you to select (efficiently) the stocks that they should recommend to their clients. For each company, they have a pair of values (x, y) where

- x is the expected annual dividends for that company's stock.
- y is a real number between -10 and 10 quantifying the risk of investing in the company (with -10 being high risk, and +10 being low risk).

So each company is represented by the point (x, y) in the plane. We will assume without loss of generality that no two companies are mapped to the same point. Given two such points, we will say that a point $q = (q.x, q.y)$ *dominates* the point $p = (p.x, p.y)$ if $q.x \geq p.x$ and $q.y \geq p.y$. That is, q lies to the right of and above p, as illustrated in picture on the left.





Clearly, a point p that is dominated by a point q is of no interest, since q is both worth more **and** less risky than p. Hence, to help the firm decide which stocks to recommend, you want to only return companies that correspond to *maximal* points: those that no other point dominates. These are the points p_3, p_4 and p_6 in the figure on the right.

1. Describe an algorithm that takes as input a set P of points, and a point q, and returns all points of P that q does not dominate. Hint: this is simple, so don't think too hard about it.

2. In order to solve this problem using a divide-and-conquer algorithm, we need to divide it into two or more subproblems. How many subproblems should we use, and is how is the input divided between these subproblems?

3. If a point is maximal in the last one of the subproblems, is it automatically maximal in P? Why or why not?

4. If a point is maximal in the first of the subproblems, is it automatically maximal in P? Why or why not?

5. Suppose that a point p in the first subproblem is dominated by one or more points in the other subproblem(s) (assuming this is possible). Name *one* point in the other subproblem(s) that is guaranteed to dominate x.

6. Using the results of the previous steps, write pseudo-code for an efficient algorithm `MaximalPoints` that takes a input a set P of points, and return a set of all maximal points of P.

7. Analyze the running time of your algorithm.

4 Bonus Problem

Can you extend the analysis of Quicksort to show that if the pivot is chosen randomly and uniformly from among the elements, then the expected running time is $\Theta(n \log n)$? Note that we cannot assume that a randomly chosen pivot is always in the range between $\lceil \frac{n}{4} \rceil$ and $\lfloor \frac{3n}{4} \rfloor$.