CPSC 320 Sample Midterm 1
October 2006

[12] 1. Short Answers

[3] a. We proved in class that algorithm `RandomizedQuickSelect` runs in $O(n)$ *expected* time. Why does this make sense, intuitively?

Solution : It makes sense because the probability of a good pivot (one that allows us to throw away at least $n/4$ elements) is 1/2. So we end up discarding 25% of the elements every other time which, if we were to draw the recursion tree, would give us a decreasing geometric series.

[3] b. In class, did we prove an $\Omega(n \log n)$ lower bound on the running time of every sorting algorithm? Explain why or why not.

Solution : No, we only proved this lower bound for *comparison sorts*, i.e. algorithms that perform no operation on the values of the elements to be sorted other than comparing two of them using $<$, $>$ or $=$.

[3] c. Can we use the limit theorem discussed in class to determine that the function $f(n)$ defined by

$$f(n) = \begin{cases} \sqrt{n} & \text{if } n \text{ is even} \\ n & \text{if } n \text{ is odd} \end{cases}$$

is in $O(n)$? Explain why or why not.

Solution : Strictly speaking, no, because $\lim_{n\to\infty} f(n)/n$ does not exist. We could however use limits to prove that $f(n) \in o(n)$ when n is even, and that $f(n) \in \Theta(n)$ when n is odd, and from that deduce that $f(n) \in O(n)$.

[3] d. A computer scientist showed how to build a decision tree for a specific algorithm $\mathcal{A}$ that decides whether or not a list contains two elements that have the same value. What does the height of this decision tree represent?

Solution : The height of the decision tree represents the worst-case number of comparisons the algorithm will make a list with n elements.

[12] 2. Prove or disprove each of the following two facts:

[6] a. $n^3 2^n \in O(3^n)$

Solution : This is true. The simplest prove uses limits:

$$\begin{aligned} \lim_{n\to\infty} n^3 2^n / 3^n &= \lim_{n\to\infty} n^3 / 1.5^n \\ &= \lim_{n\to\infty} 3n^2 / (1.5^n \log_e 1.5) \end{aligned}$$

$$\begin{aligned} &= \lim_{n\to\infty} 6n/(1.5^n \log_e^2 1.5) \\ &= \lim_{n\to\infty} 6/(1.5^n \log_e^3 1.5) \\ &= 0 \end{aligned}$$

and so $n^3 2^n \in o(3^n)$, which implies that $n^3 2^n \in O(3^n)$.

[6] b. Let f, g, h and k be functions from $\mathbf{N}$ into $\mathbf{R}^+$. If $f(n) \in \Omega(g(n))$ and $h(n) \in \Omega(k(n))$, then $f(n)h(n) \in \Omega(g(n)k(n))$.

Solution: This is also true. Since $f(n) \in \Omega(g(n))$, there are constants n_0, $c > 0$ such that $f(n) \geq cg(n)$ for every $n \geq n_0$. Similarly, since $h(n) \in \Omega(k(n))$, there are constants n_0', $c' > 0$ such that $h(n) \geq c'k(n)$ for every $n \geq n_0'$. Hence, for every $n \geq \max\{n_0, n_0'\}$, $f(n)h(n) \geq (cg(n))(c'k(n)) = (cc')(g(n)k(n))$.

[11] 3. Consider the recurrence relation

$$T(n) = \begin{cases} 4T(n-1) + 4^n & \text{if } n \geq 2 \\ 1 & \text{if } n = 1 \end{cases}$$

[3] a. Explain briefly why we can not use the Master theorem directly to solve this recurrence relation.

Solution: Because the Master theorem can only be applied to a recurrence relation of the form $T(n) = aT(n/b) + f(n)$. The term $T(n-1)$ is not in the form $T(n/b)$.

[8] b. Prove upper and lower bounds on the function $T(n)$. Your grade will depend on the quality of the bounds you provide (that is, showing that $T(n) \in \Omega(1)$ and $T(n) \in O(100^n)$, while true, will not give you many marks).

Solution: The simplest way to solve the recurrence is to use a recursion tree. The i^{th} level of the tree contains 4^i nodes, each of which performs 4^{n-i} work (for $i = 0, 1, \ldots n$). Hence the total amount of work done is exactly $(n+1)4^n$, which is in $\Theta(n4^n)$.

[10] 4. In this question, we consider the problem of computing the smallest gap between two elements in an array of real numbers. That is, we want to find the smallest difference between two distinct elements of the array. For instance, for the array $(3.8, 8.1, 1, 6.2, 2.5)$, the smallest gap is 1.3 because $3.8 - 2.5 = 1.3$, and no other pair of elements has a difference smaller than 1.3.

[7] a. Design a divide and conquer algorithm to compute the smallest gap in the array that is passed as parameter. Your grade will depend on both the elegance and the efficiency of your algorithm.

Solution: Here are two possible solutions. The first solution is very straightforward, but has a worst-case running time of $O(n^2)$ – it was worth 6/7.

```
Algorithm SmallestGap(A, p, r)
  if (p = r)
      return +∞

  q ⟵ (p + r)/2
  g1 ⟵ SmallestGap(A, p, q)
  g2 ⟵ SmallestGap(A, q+1, r)

  gap = min(g1, g2)
  for i ⟵ p to q do
      for j ⟵ q+1 to r do
          gap = min(gap, abs(A[i] - A[j]))

  return gap
```

A better solution would be to sort the array (using mergesort, a divide and conquer algorithm) and then either go through the array linearly checking adjacent pairs of elements, or find the minimum using divide and conquer also.

```
Algorithm SmallestGap(A, p, r)
  MergeSort(A, p, r)
  return SmallestGapHelper(A, p, r)

Algorithm SmallestGapHelper(A, p, r)
  if (p = r)
      return +∞

  q ⟵ (p + r)/2
  g1 ⟵ SmallestGap(A, p, q)
  g2 ⟵ SmallestGap(A, q+1, r)

  return min(g1, g2, A[q+1] - A[q])
```

[3] b. Analyze the running time of the algorithm you described in your answer to part (a).

Solution : The first algorithm has a running time described by the recurrence

$$T(n) = \begin{cases} 2T(n/2) + \Theta(n^2) & \text{if } n \geq 2 \\ 1 & \text{if } n = 1 \end{cases}$$

and by case 3 of the Master theorem, it runs in $\Theta(n^2)$ time. The second algorithm runs in time $n \log n + S(n)$ where

$$S(n) = \begin{cases} 2T(n/2) + \Theta(1) & \text{if } n \geq 2 \\ 1 & \text{if } n = 1 \end{cases}$$

and hence runs in time $\Theta(n \log n + n) = \Theta(n \log n)$.