CPSC 320 Sample Final Examination
December 2010

[10] 1. Answer each of the following questions with *true* or *false*. Give a *short* justification for each of your answers.

[5] a. $6^n \in O(5^n)$

Solution: This is false:

$$\begin{aligned} \lim_{n\to+\infty} \frac{6^n}{5^n} &= \lim_{n\to\infty} \left(\frac{6}{5}\right)^n \\ &= +\infty \end{aligned}$$

which means that $6^n \in \omega(5^n)$. Therefore $6^n \notin O(5^n)$.

[5] b. $1.5^n + n^2 \in O(1.5^n + n \log n)$

Solution: This is true, since for $n \geq 13$ we have $n^2 < 1.5^n$, which means that $1.5^n + n^2 \leq 1.5^n + 1.5^n = 2 \cdot 1.5^n < 2 \cdot (1.5^n + n \log n)$. Hence take $c = 2$ and $n_0 = 13$.

[18] 2. Short Answers

[3] a. Why is it important to know that a problem is NP-Complete?

Solution: Because then we know not to waste time trying to find an efficient algorithm for it that always returns the correct answer.

[3] b. Explain why the following statement holds: "Finding the median of an unordered array is as difficult as finding the i^{th} order statistics of that array, for an arbitrary i". Hint: think of the implementations of algorithm `RandomizedQuickSelect`

Solution: In order to find the median of the array, we will most likely end up trying to find the i^{th} smallest element of a subset of m elements, where $i \neq \lceil m/2 \rceil$. In that sense, in order to be able to find the median, we need to be able to find an arbitrary order statistics. Thus finding the median is not any easier.

[3] c. What is the main advantage of `RandomizedQuickSelect` over `Select1`?

Solution: `RandomizedQuickSelect` performs well on average on every possible input, whereas `Select1` performs very well (always) on some inputs, and very badly (always) on other inputs.

[3] d. The Prim-Jarník algorithm associates a cost with each vertex that has not yet been added to the tree it constructs. What does this cost represent?

Solution: It represents the cost of the cheapest edge that connects the vertex to the part of the minimum spanning tree that has already been constructed.

[3] e. Why does the Gale-Shapley (Stable Matching) algorithm discussed in class terminate after at most n^2 iterations?

Solution: Because at every step a woman proposes to a man she has never proposed to. There are only n^2 possible couples, and hence the loop could not iterate more than n^2 times without repeating a proposal.

[3] f. When do we use amortized analysis?

Solution: When we are trying to prove a good upper-bound on the worst-case running time of a sequence of n operations.

[9] 3. A Computer Science researcher has designed a new data structure called a *sheap* that supports operations `insert`, `findMinMax` and `extractMinMax`, and decides to use amortized analysis to determine the worst-case running time of a sequence of n operations on an initially empty sheap. She defines a potential function Φ for sheaps, such that $\Phi(S) \geq 0$ for every sheap S, and such that $\Phi(S) = 0$ if the sheap S is empty. After analyzing the running time of the three operations, she has learned that

- An `insert` operation on a sheap with n elements takes time $\log n$, and increases the sheap's potential by 2.
- A `findMinMax` operation on a sheap with n elements takes time $x + \sqrt{n}$, where x is the number of elements examined by the operation. The potential of the sheap goes down by $x - 1$.
- A `extractMinMax` operation starts by performing a `findMinMax` operation. The remainder of the `extractMinMax` operation takes time $\log n$, and decreases the sheap's potential by 1.

Give as tight a bound as possible on the worst-case running time of a sequence of n operations on an initially empty sheap.

Solution: The amortized cost of `insert` is $\log n+2$, the amortized cost of `findMinMax` is $x + \sqrt{n} - (x-1) = \sqrt{n}+1$ and the amortized cost of `extractMinMax` is $\sqrt{n}+1+\log n - 1 = \sqrt{n} + \log n$. We thus get an upper bound of

$$\sum_{i=1}^{n} \max\{\log n + 2, \sqrt{n}+1, \sqrt{n}+\log n\} \in O(n\sqrt{n})$$

on the worst-case running time of a sequence of n operations.

[9] 4. For each of the following recurrence relations, determine whether or not the Master Theorem discussed in class can be used. If it can be used, apply it to derive the solution of the recurrence relation using O notation. If the Master Theorem can not be used, explain why briefly.

[3] a. $T(n) = \begin{cases} 2T(\lfloor\sqrt{n}\rfloor) + n & \text{if } n \geq 2 \\ 1 & \text{if } n \leq 1 \end{cases}$

Solution: No, because the term inside the $T()$ is not in the form n/b where b is a constant.

[3] b. $T(n) = \begin{cases} 9T(n/3) + 2n^2 & \text{if } n \geq 3 \\ 1 & \text{if } n \leq 2 \end{cases}$

Solution: Yes: this is case 2 of the Master theorem, and $T(n) \in \Theta(n^2 \log n)$.

[3] c. $T(n) = \begin{cases} 4T(\lfloor n/2 \rfloor) + n^{2+\text{odd}(n)} & \text{if } n \geq 2 \\ 1 & \text{if } n \leq 1 \end{cases}$ where $\text{odd}(n) = \begin{cases} 1 & \text{if } n \text{ is odd} \\ 0 & \text{if } n \text{ is even} \end{cases}$

Solution: No: this recurrence is sometimes in case 2 (when n is even) and sometimes in case 3 (when n is odd), so the theorem can not be used.

[12] 5. Consider the following function:

```
Algorithm HappyNewYear(A, p, r)
//
// A is an array, p and r are positions in the array.
//
mid ← ⌊ (p + r)/2 ⌋
x ← Christmas(A[p]) + Christmas(A[mid]) + Christmas(A[r])
if (n ≥ 2) then
    x ← x + HappyNewYear(A, p+1, r-1)
    x ← x + HappyNewYear(A, p+1, r-1)
    x ← x + HappyNewYear(A, p+1, r-1)
    x ← x + HappyNewYear(A, p+1, r-1)
return x
```

[4] a. Let $C(n)$ be the number of times that function `Christmas` will be called during the execution of the call `HappyNewYear(A,p,r)`, where $n = r - p + 1$. Write a recurrence relation for $C(n)$.

Solution: $C(n) = \begin{cases} 3 + 4C(n-2) & \text{if } n \geq 2 \\ 3 & \text{if } n < 2 \end{cases}$

[8] b. Prove using the guess and test method that the solution of the recurrence relation you gave in part (a) is in $O(2^n)$.

Solution: Let us prove that $C(n) \leq c2^n - x$ for some constants c and x that we will determine later (guessing that $C(n) \leq c2^n$ would not work as we

would end up with a +3 term that we can not get rid of). We will prove the guess using the strong form of mathematical induction.

We start by the induction step. Suppose that $C(i) \leq c2^i - x$ for $i = 0, 1, \ldots, n-1$. Consider now $C(n)$:

$$\begin{aligned} C(n) &= 3 + 4C(n-2) \\ &\leq 3 + 4\left(c2^{n-2} - x\right) \\ &\leq 3 + 4c2^{n-2} - 4x \\ &\leq 3 + c2^n - 4x \\ &\leq (3 - 3x) + c2^n - x \end{aligned}$$

Let us choose $x = 1$. The term $3 - 3x$ is then equal to 0 and can be discarded, completing the induction step.

Now for the base cases: we need $C(0) = 3 \leq c2^0 - x = c - x$, and $C(1) = 3 \leq c2^1 - x = 2c - x$. If we choose $c = 4$, then both conditions hold.

Therefore $C(n) \leq 4 \cdot 2^n - 1$ for every $n \geq 0$.

[10] 6. The Disjoint Sets data structure

a. When we proved that the amortized cost of a `find(N)` operation is in $O(\log^* n)$, the nodes on the path from N to its root where divided into two types. What were these, and how was the cost of traversing these nodes accounted for?

Solution: First, there were nodes who rank is in the same interval as their parent's rank. We accounted for the cost of traversing them using the potential stored in the data structure (the potential decreased by 1 for each of them). There were also nodes whose rank is in a different interval from their parent's rank. The cost of traversing these nodes is what gave us the $O(\log^* n)$ amortized cost (that is, we had to "pay" for these out of our own pocket).

b. How did we prove that the total infusion of potential into the data structure, which happens when a `union` operation is performed, did not add more than $n \log^* n$ units of potential?

Solution: We used the fact that no more than $n/2^k$ nodes have rank $\geq k$ to get a bound on the number of nodes that are allocated 2^t units of potential, and hence get a bound on the maximum possible potential increase.

[15] 7. The CEO of a software company wants to keep his developers happy by giving them a bonus, but does not want to spend too much money. He thus wants to select as few developers as possible (these will get a bonus, and be happy), chosen so that every other developer likes at least one of those that get a bonus (and hence will be happy for him/her).

The CEO's problem can be modeled using a graph $G = (V, E)$: each node in the graph is a developer, and an edge (u, v) means that developer u likes developer v.

The CEO is looking for a minimum subset W of the set V of vertices, where for every vertex $u \notin W$, there is an edge (u, w) where $w \in W$. This is called a *minimum dominating set* for the graph G.

[9] a. Write a greedy algorithm that finds a subset W of V (it does need to be the subset with the fewest elements possible) with that property. Your mark will depend in part on the criterion you use to decide which vertex to add to W. You do not need to give pseudo-code, but you should indicate what data structure you are using to store the vertices that your algorithm has not dealt with yet.

Solution: We store in each node N whether or not developer N is happy. The algorithm then proceeds as follows:

```
while at least one vertex is not happy do
  for each vertex v of G
      set count(v) to 0 if v is happy, and 1 otherwise

  for each edge (v1, v2) of G
      if v1 is not happy then
        increment count(v2)

  let v be the vertex with maximum count
  mark v happy
  add v to W

  for each edge (v1, v) of G
      mark v1 happy
endwhile
```

[3] b. What is the running time of the algorithm you described in part (a)?

Solution: This algorithm will run in time $O(|V||E|)$ time.

[3] c. The minimum dominating set problem does not satisfy the greedy choice property. Knowing this, what can you conclude about the algorithm you gave in part (a)?

Solution: It does not always return an optimal solution (that is, the smallest W).

[17] 8. Bill and Simon have been arguing about where to have lunch after the CPSC 320 final examination, and decide to solve the decision by playing a sequence of Poker games. The first person to win n games gets to choose where they will have lunch. Simon is a slighly better poker player, and has a 51% chance of winning any individual Poker game.

Let $P(i, j)$ be the probability that Bill will be the first person to win n games, given that he only needs to win another i games (that is, Bill has won $n - i$ games so far), and that Simon only needs to win another j games (that is, she has won $n - j$ games so far). It can be proved that

$$P(i, j) = 0.49P(i - 1, j) + 0.51P(i, j - 1) \tag{1}$$

Bill is worried about his chances of winning, and asks you to use dynamic programming to compute the probability $P(n, n)$ that he will win n games before Simon does.

[3] a. State how big a table you need to answer Bill's question using dynamic programming, and the meaning of each entry in the table.

Solution: Both i and j will take values from 0 to n, and so we need a $n+1$ by $n+1$ table.

[3] b. Give one possible order that you can use to compute the entries in the table from part (a).

Solution: We can compute them by increasing value of j, and for each j by increasing value of i.

[3] c. List all of the base cases that will be needed when you are computing the entries in the table from part (a). That is, list the cases where you can not use equation (1) to compute $P(i, j)$, and the value of $P(i, j)$ for each of them.

Solution: If $i = 0$ and $j > 0$ then $P(i, j) = 1$ (Bill has already won). If $i > 0$ and $j = 0$ then $P(i, j) = 0$ (Bill has already lost).

[8] d. Write pseudo-code for an algorithm that uses dynamic programming to determine the probability that Bill will win n Poker games before Simon does.

Solution:

```
for i ← 1 to n do
    P[i,0] ← 0

for j ← 1 to n do
    P[0,j] ← 1
    for i ← 1 to n do
        P[i,j] = 0.49*P[i-1,j] + 0.51*P[i,j-1]

return P[n,n]
```