CPSC 320 Sample Final Examination
April 2009

[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. What factor(s) does the average-case running time of `SkipListSearch` on a given skip list (averaged over all keys one could search for) depend on?

Solution : It depends only on the levels of the nodes in the skip list (in other words, it does not depend on the order of insertion, or on whether or not nodes were deleted at some point).

[3] b. True or false: for small values of n (for instance $n \leq 800$ or so), algorithm DeterministicSelect is slower than the naive $O(n\log n)$ algorithm (the one that sorts the array using quicksort or mergesort, and then returns the requested element in constant time).

Solution : This is true. The constant hidden by the O notation for algorithm DeterministicSelect is very large.

[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. Name one reason why the choice of a random level for each node in a Skip List helps us achieve a good average case running time.

Solution : Because we end up with approximately $1/p$ more nodes on level $i-1$ than we had on level i, and these nodes are more or less evenly distributed between the nodes on the level i list.

[3] e. What is the main difference, in terms of the running time of their operations, between a binomial heap and a Fibonacci heap?

Solution : Every operation on a binomial heap runs in $O(\log n)$ worst-case time. Most operations on a Fibonacci heap run in $O(1)$ time (amortized, for `decrease-key`), while `extract-min` and `delete` run in $O(\log n)$ amortized time (with a $\Theta(n)$ worst-case time).

[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. In class (or in previous courses you took), we learned about the following algorithms, all of which take an array A as parameter.

(a) DeterministicSelect
(b) Heapsort
(c) Insertion sort
(d) RandomizedQuicksort
(e) RandomizedQuickSelect
(f) Quicksort

For each of the statements below, list *all* of the algorithms listed above for which the statement is true (you do not need to justify your answer(s)).

[3] a. If we change the ordering of the elements in the array A, then the algorithm's running time may change by more than a constant factor (even if everything else remains unchanged).

Solution : First, let us agree that "everything else remaining unchanged" includes the random values returned by the random number generator. With that in mind, the answer is Insertion Sort (n versus n^2), RandomizedQuicksort (different positions for the values will give us different pivots), RandomizedQuickSelect, and Quicksort (same reasons).

[3] b. Its average-case running time $T_a(n)$ is much better than its worst-case running time $T_w(n)$, that is, $T_a \in o(T_w)$.

Solution : RandomizedQuicksort, RandomizedQuickSelect and Quicksort.

[3] c. The amount of storage used when the algorithm is executed, not counting the input array A, is in $o(n)$.

Solution : All of them can be implemented this way, although you have to be careful with the implementation of DeterministicSelect, and *very* careful with the two quicksort variants.

[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.

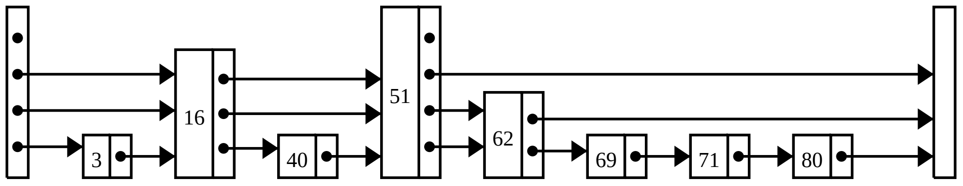
Therefore $C(n) \leq 4 \cdot 2^n - 1$ for every $n \geq 0$.

[20] 6. Skip Lists

[4] a. When a new key is inserted in a skip list, we choose the level of its node randomly, where for each i the probability that the node has level $\geq i$ is p^{i-1}. Explain briefly how this is done. Recall that p is a constant between 0 and 1 that is chosen beforehand (it is easier to think of $p = 1/2$).

Solution: We throw a $1/p$ sided dice until we get a "1" (which occurs with probability p), and use the number of throws as the level of the new node.

[3] b. Consider now the following skip list:



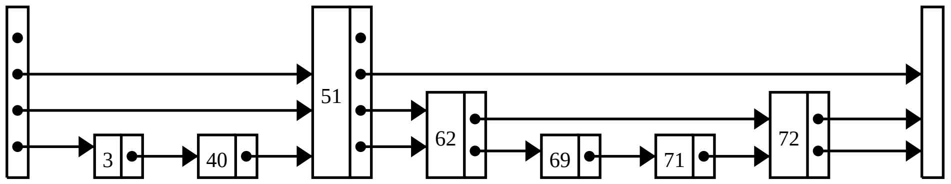
Show the pointers *followed* by the Search algorithm discussed in class, when the key to search for is 73. Assume that the current value of `MaxLevel` is 3.

Solution: SkipListSearch will follow: the level-3 pointer from the head node to the "16" node, the level-3 pointer from the "16" node to the "51" node, the level-2 pointer from the "51" node to the "62" node, the level-1 pointer from the "62" node to the "69" node, and the level-1 pointer from the "69" node to the "71" node.

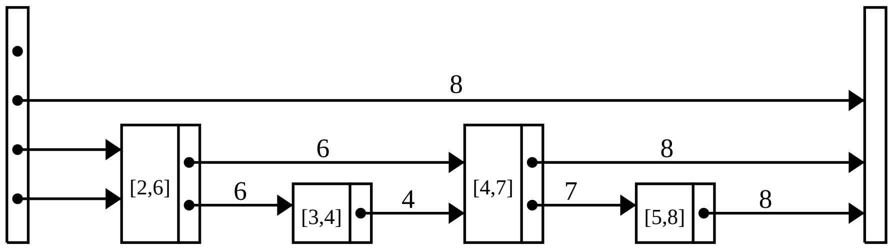
[5] c. Show the skip list obtained after the following sequence of operations has been performed on the skip list from part (b): inserting key 44 with level = 3,

deleting key 80, deleting key 16, inserting key 72 with level = 2, and deleting key 44. Hint: it is faster if you do them all at once.

Solution:



[8] d. Suppose now that we have a skip list whose keys are intervals, sorted according to their left endpoint, like the following skip list (its current MaxLevel is 3):



We have augmented that skip list by storing in each pointer P the maximum right endpoint of the intervals stored between the node containing P and the node immediately preceding the node the pointer points to. For instance, the second pointer at level 2 contains the value 6 because 6 is the maximum right endpoint amongst the two intervals $[2, 6]$ and $[3, 4]$

Explain how you would update the values associated with the pointers when a new key is inserted.

Solution: After inserting the new node N, we need to update

- The pointers that bypass the new node (at levels higher than N's): it is just a matter of taking the maximum between the value currently associated with that pointer, and the right endpoint of the interval stored in N.
- Pointers that point to N, or are contained in N: we do this one level at a time, starting at level 1. Values at level 1 are trivial to compute. For the other levels, we recompute the value on a pointer P at level i by traversing the pointers at level $i - 1$, starting from the node containing P up to the node that P points to, and taking the maximum value stored in one of them.

[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]
```