York University
COSC 3101 Fall 2004 – Midterm (Nov 4)
Instructor: Jeff Edmonds

1. $2 \times 9 = 18$ marks: Compute the solution and tell which rules that you use.

   (a) $\sum_{i=0}^{n} 3^i \times i^8 = \Theta($
   Type of sum:

   - Answer: $\Theta(3^n \times n^8)$, Geometric.

   (b) $\sum_{i=0}^{n} 300i^2 \log^9 i + 7\frac{i^3}{\log^2 i} + 16 = \Theta($
   Type of sum:

   - Answer: $\Theta(\frac{n^4}{\log^2 n})$, Arithmetic.

   (c) $\sum_{i=0}^{n} \sum_{j=0}^{n} i^2 j^3 = \Theta($

   - Answer: $\sum_{i=1}^{n} \sum_{j=0}^{n} i^2 j^3 = \sum_{i=1}^{n} i^2[\sum_{j=0}^{n} j^3] = \sum_{i=1}^{n} i^2\Theta(n^4) = \Theta(n^4)[\sum_{i=1}^{n} i^2] = \Theta(n^4)\Theta(n^3) = \Theta(n^7)$

   (d) $7 \cdot 2^{3 \cdot n^5} = 3^{\Theta(n^5)}$ **True** **False**

   - Answer: True

   (e) What is the formal definition of $f(n) = n^{\Theta(1)}$?

   - Answer: $\exists c_1 > 0, c_2 > 0, n_0,\ \forall n \geq n_0,\ n^{c_1} \leq f(n) \leq n^{c_2}$.

   (f) $n^2 - 100n \in o(n^2)$ **True** **False**

   - Answer: False

   (g) If $T(n) = 8T(\frac{n}{4}) + \Theta(n)$. Then $T(n) = \Theta($

   - Answer: Because $\frac{\log a}{\log b} = \frac{\log 8}{\log 4} = \frac{\log_2 8}{\log_2 4} = \frac{3}{2} = 1.5 > 1 = c$, the technique concludes that time is dominated by the base cases and $T(n) = \Theta(n^{\frac{\log a}{\log b}}) = \Theta(n^{1.5})$

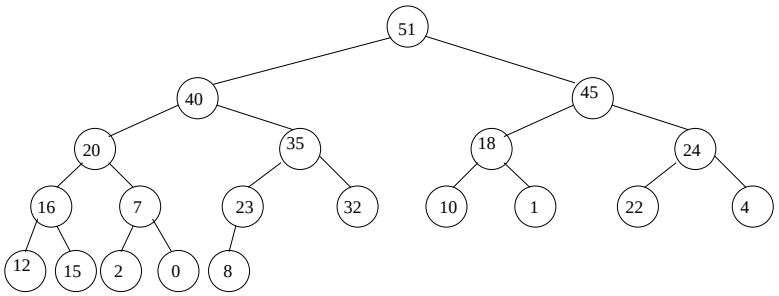
   (h) The solution of the recurrence $T(n) = 5T(\frac{n}{\sqrt{5}}) + n^2 \log(n)$ is
   $T(n) = \Theta($

   - Answer: Because $\frac{\log a}{\log b} = \frac{\log_{\sqrt{5}} 5}{\log_{\sqrt{5}} \sqrt{5}} = \frac{2}{1} = 2 = 2 = c$, the technique concludes that time is dominated by the base cases and $T(n) = \Theta(f(n) \log n) = \Theta(n^2 \log^2 n)$.

   (i) If $T(n) = 8T(n-5)$. Then $T(n) = \Theta($

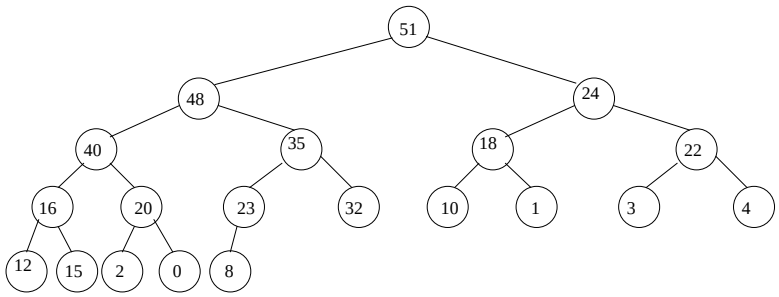
   - Answer: $T(n) = 8T(n-5) = 8^2T(n-2\times 5) = 8^3T(n-3\times 5) = 8^iT(n-i\times 5) = \Theta(8^{i/5})$.

2. 4 marks: Consider the following priority queue implemented as a heap. Consider a reasonable algorithm that changes the prioity of a node. (Its input includes a pointer to the node). Give the resulting heap after changing the priority 7 node to have priority 48 and and the priority 45 node to 3.



- Answer: Question cut from midterm: "In three or four sentences, describe the algorithm for changing the priority of a node." Ans: If the priority is increased, bubble it up by repeatedly comparing it to its parent and swapping if bigger. If the priority is decreased, bubble it down by repeatedly comparing it to both its children and swapping if smaller with the bigger of the two.

- Answer:



3. 6 marks: Considering Programs.

algorithm $Eg_1(n)$
⟨*pre−cond*⟩: n is an integer.
⟨*post−cond*⟩: Prints "Hi"s.

```
begin
    i = 0
    put "Hi"; put "Hi"; put "Hi"; put "Hi"
    loop i = 1 ... n
        put "Hi"; put "Hi"; put "Hi"
        loop j = 1 ... n
            put "Hi"; put "Hi"
        end loop
    end loop
end algorithm
```

algorithm $Eg_2(n)$
⟨*pre−cond*⟩: n is an integer.
⟨*post−cond*⟩: Prints $T(n)$ "Hi"s.

```
begin
    if( n ≤ 1) then
        put "Hi"
    else
        put "Hi"; put "Hi"; put "Hi"
        Eg2(n/4)
        Eg2(n − 5)
    end if
end algorithm
```

(a) Give the exact time complexity (running time) of Eg_1.

- Answer: $Size = \log n = s$. $n = 2^s$. $Time = 2n^2 + 3n + 4 = 2 \cdot 2^{2s} + 3 \cdot 2^s + 4$.

(b) Give a recurrence relation for the running time of Eg_2. Do not solve it.

- Answer: $T(1) = 1$; $T(n) = T(\frac{n}{4}) + T(n-5) + 3$.

4. 8 marks: Briefly describe and contrast the difference between a "More of the Input" loop invariant and a "More of the Output" loop invariant. Give an example of each along with a picture.

- Answer: A loop invariant of the first type says "I currently have a solution for the first i objects." An example is insertion sort, whose loop invariant is that the first i values from the input are sorted.

  A loop invariant of the first type says "I have constructed i correct pieces of the output so far." An example is selection sort, whose loop invariant is that the first i values from the output (i.e. the smallest) are in place.

5. 20 marks: Iterative Algorithms: You are now the professor. Which of the steps to develop an iterative algorithm did the student fail to do correctly in Eg_3? How?

algorithm $Eg_3(I)$

$\langle pre-cond \rangle$: I is an integer.

$\langle post-cond \rangle$: Outputs $\sum_{j=1}^{I} j$.

```
begin
    s = 0
    i = 1
    while( i ≤ I )
        ⟨loop−invariant⟩: Each iteration adds the next
            term giving that s = Σ_{j=1}^{i} j.
        s = s + i
        i = i + 1
    end loop
return(s)
end algorithm
```

- Answer: 2×10 marks: There are a number of problems.
  (a) A loop invariant is to be a picture of the current state and NOT say what the iteration does.
  (b) The loop invariant is not established correctly. With $i = 1$, the loop invariant requires $s = \sum_{j=1}^{1} j = 1$ not $s = 0$. $s = 0$ and $i = 0$ would be better.
  (c) The loop invariant is not maintained correctly. Let s' and i' be the values of s and i when at the top of the loop. Let s'' and i'' be the values after going around again. The LI gives that $s' = \sum_{j=1}^{i'} j$. The code gives that $s'' = s' + i'$ and $i'' = i' + 1$. Together this gives that $s'' = (\sum_{j=1}^{i'} j) + i'$. This is not $\sum_{j=1}^{i'+1} j$ as required because i' is being added in twice. $i' + 1$ should be added in order to maintain the LI.
  (d) The exit condition is not very well stated. An equivalent and easier to see exit condition would be "exit when $i > I$".
  (e) The exit condition, $i > I$, and the LI, $s = \sum_{j=1}^{i} j$, together do not give the post condition. Instead, they give that $s = \sum_{j=1}^{I+1} j$ is returned.

- Answer: Deleted Question: Does the program for Eg_3 work? Ans: It happens to work. But the loop invariant is wrong.

6. 10 marks: Consider my solution to Q3 in Assignment 2, which finds for each pair of nodes $u, v \in V$ of a graph a path from u to v with the smallest total weight from amongst those paths that contains exactly k edges.

   (a) 4 marks: What are the steps for maintaining the loop invariant.

   - Answer: $w_{u,v,i+j} = \min_b(w_{u,b,i} + w_{b,v,i})$.

   (b) 4 marks: What needs to be changed so that instead of the path having EXACTLY k edges, it has k or fewer edges?

   - Answer: The loop invariant is changed so that $w_{u,v,i}$ is the weight of the smallest weighted path from u to v containing i or fewer edges. Similarly j.
     It turns out that the steps to maintain the loop invariant do not need to change at all. Any path from u to v with $i + j$ or fewer edges (and at least 2 edge) consists of a path from u to some b with i or fewer edges and then from b to v with j or fewer edges. The path might also contain only one edge. But then if we let $b = u$, then the path from u to v is a path of length zero ($\leq i$) from u to $u = b$ followed by a path of one ($\leq j$) edge from $u = b$ to v. If $u = v$, then the path to lenght of zero ($\leq i + j$) is a path of length zero ($\leq i$) from u to $u = b$ plus a path of length zero ($\leq j$) from $u = b$ to $u = v$. It all works.
     The only difference is when we exablish the loop invariant with $i = 1$, we need to consider paths of lenght zero from u to u. This can be done by assuming that each node has a self loop with weight $w_{u,u} = 0$.

(c) 2 marks: What needs to be changed so that for every u and v it outputs the shortest over all path. (Assume the weights are all positive.)

- Answer: When all the weights are positive, no smallest weighted path will contain a loop and hence none will be longer than $n-1$ edges. Hence, the above problem is solved by setting $k = n-1$ (or even better a power of 2 bigger at least $n-1$.

7. 21 marks: Recursion:

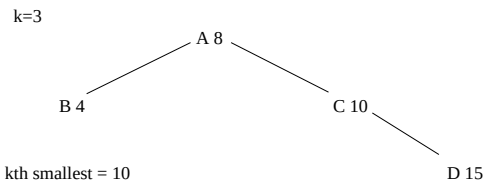
(a) 4 marks: With Iterative algorithms, Jeff is obsessed with Loop Invariants. Describe in full the scenario that he is obsessed with regarding Recursion. (How is Recursion abstracted?)

- Answer: Friends & Strong Induction: The easiest method is to focus separately on one step at a time. Suppose that someone gives you an instance of the computational problem. You solve it as follows. If it is sufficiently small, solve it yourself. Otherwise, you have a number of friends to help you. You construct for each friend an instance of the same computational problem that is *smaller* then your own. We refer to these as *subinstances*. Your friends magically provide you with the solutions to these. You then combine these *subsolutions* into a solution for your original instance. I refer to this as the *friends* level of abstraction. If you prefer, you can call it the *strong induction* level of abstraction and use the word "recurse" instead of "friend." Either way, the key is that you concern yourself only about your stack frame. Do not worry about how your friends solve the subinstances that you assigned them. Similarly, do not worry about whomever gave you your instance and what he does with your answer. Leave these things up to them.

(b) 3 marks: Which are the general shapes of trees that you should check your program on. (Binary trees)

- Answer:
  i. General tree with big left and right subtrees.
  ii. General tree with big left subtree and empty right.
  iii. General tree with empty left subtree and big right.
  iv. Empty tree.

(c) 9 marks: Finding the $\boldsymbol{k^{th}}$ Smallest Element: Write a recursive program, which given a binary search tree and an integer $\boldsymbol{k}$ returns the $\boldsymbol{k^{th}}$ smallest element from the tree. (Code is fine.)



- Answer:

algorithm $Smallest(tree, k)$
⟨pre−cond⟩: $tree$ is a binary search tree and $k > 0$ is an integer.
⟨post−cond⟩: Outputs the k^{th} smallest element s and the number of elements n.
begin
  if($tree = emptyTree$) then
    result($\langle NotPossible, 0\rangle$)
  else
    $\langle s_l, n_l\rangle = Smallest(leftSub(tree), k)$
    % There are $n_l + 1$ nodes before the right subtree
    $\langle s_r, n_r\rangle = Smallest(rightSub(tree), k - (n_l + 1))$
    $n = n_l + 1 + n_r$
    if($k \in [1..n_l]$)then
      $s = s_l$
    elseif($k = n_l + 1$)then
      $s = root(tree)$

```
            elseif( k ∈ [n_l + 2..n] )then
                s = s_r
            else then
                s = OutOfRange
            endif
            result( ⟨s, n⟩ )
        end if
    end algorithm
```

(d) 2 marks: Give the recurrence relation and the running time for your program when the input tree is completely balanced.

- Answer: $T(n) = 2T(n/2) + 1 = \Theta(n)$.

(e) 3 marks: Prove that no program can solve the problem by more than a constant factor faster.

- Answer: If $\Theta(n)$ nodes are in the left subtree and the answer is in the right, then the nodes in the left need to be counted and this will take at least time $\Theta(n)$.

8. 13 marks: Dijkstra's Algorithm:

(a) 4 marks: Give the full loop invariant for Dijkstra's Algorithm. Include the definition of any terms you use.

- Answer:

  The Loop Invariant:

  LI1: For each handled node v, the values of $d(v)$ and $\pi(v)$ give the shortest length and a shortest path from s (and this path contain only handled nodes).

  LI2: For each of the unhandled nodes v, the values of $d(v)$ and $\pi(v)$ give the shortest length and path from among those paths that have been *handled*.

  Definition of A Handled Path: We say that a path has been handled if it contains only handled edges. Such paths start at s, visit as any number of handled nodes, and then follow one last edge to a node that may or may not be handled.

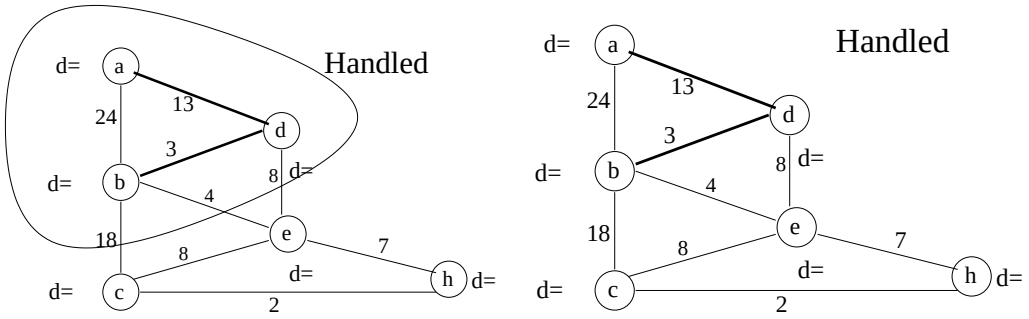
(b) 2 marks: What is the exit condition for Dijkstra's Algorithm?

- Answer: All the nodes have been handled

(c) 2 marks: Prove that the post condition is obtained.

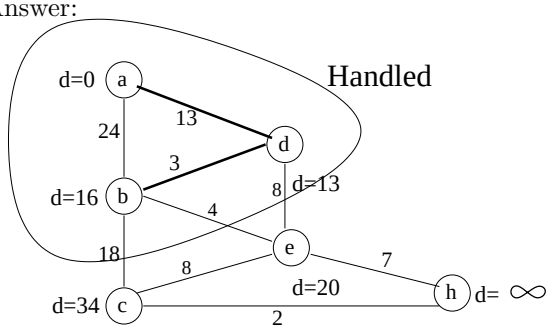
- Answer: When the loop exits, both the exit condition and the LI are true. The exit condition gives that all nodes are handled. The LI gives that we have the correct answer for all handled nodes. It follows that we have the correct answer for all nodes.

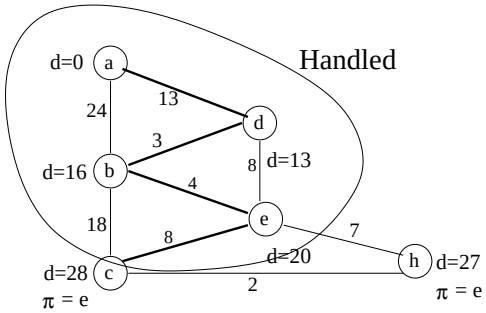
(d) 3 marks: Consider a computation of Dijkstra's algorithm on the following graph when the circled nodes have been handled. The start node is a. On the left, give the current values of d.



(e) 2 marks: On the right, change the figure to take one step in Dijkstra's algorithm. Include as well any πs that change.

- Answer:


Handled
d=0
a
13
24
d
3
d=16
b
8
d=13
4
18
e
7
8
d=20
h
d= ∞
d=34
c
2


Handled
d=0
a
13
24
d
3
8
d=13
d=16
b
4
18
e
7
8
d=20
h
d=27
d=28
c
2
π = e
π = e