

## Midterm Exam

Name: \_\_\_\_\_

SS Number: \_\_\_\_\_

**Instructions:** Answer all questions.

- (10) We want to implement multitasking on a computer with a single processor. In other words, we want to create the illusion of processing several jobs simultaneously by letting the processor compute for a time on one process, then switch to another process and compute a bit, and so on, until all jobs are finished. Let a *node* be a data element containing all information about a suspended process including information needed to resume that process. If we want to be fair to all jobs, what data structure should we should use to hold nodes?

*Answer:* a Queue

If we want to be diabolically unfair, what data structure should be used?

*Answer:* a Stack

- (20) The following code is used to sort the integers of the array `lst` where, initially, `lst[0]=9`, `lst[1]=3`, `lst[2]=6`, `lst[3]=1`, `lst[4]=10`, `lst[5]=8`.

```
void SortOfSorter(int *lst, int a, int b) {
    if (a >= b) return;

    SortOfSorter(lst, a, (a+b)/2);
    SortOfSorter(lst, (a+b)/2+1, b);
    visit(lst, a, (a+b)/2, b);
}
```

The function `visit` takes as input an array `lst` of integers where  $lst[a] \leq lst[a+1] \leq \dots \leq lst[(a+b)/2]$  and  $lst[(a+b)/2+1] \leq lst[(a+b)/2+2] \leq \dots \leq lst[b]$ , and rearranges some or all elements of `lst` so that  $lst[a] \leq lst[a+1] \leq \dots \leq lst[b]$ . Show the contents of `lst` after each call to `visit` is completed by filling in the boxes on the next page, one row of boxes per call to `visit`. Fill the left column, then the right column, if needed, from top to bottom.

1	3	9	6	1	10	8	9						
2	3	6	9	1	10	8	10						
3	3	6	9	1	10	8	11						
4	3	6	9	1	8	10	12						
5	1	3	6	8	9	10	13						
6							14						
7							15						
8							16						

Consider the set  $O = \{o_1, o_2, \dots, o_n\}$  of  $n$  colorable objects. Call a set of two *distinct* objects, say  $\{o_i, o_j\}$  a *pairing*. Suppose you are given a red box containing a bunch of pairings of objects from  $O$  but without duplicates (all pairings are distinct). The remaining questions are based on the following algorithm:

Color all objects blue.

Arbitrarily choose, but do not remove, a pairing in the red box.

Let the left object of the pairing be referred to as  $o$ .

Color  $o$  brown.

Repeat the following until there are no brown objects:

Open and empty a black box.

Put all currently brown objects into the black box.

Repeat the following until the black box is empty:

Remove object  $x$  from the black box.

Color object  $x$  yellow.

Repeat the following until no pairing in the red box has  $x$  on the left:

Remove a pairing  $p$  from the red box with  $x$  on the left.

If the right object in  $p$  is blue, color it brown.

3. (5) What is so special about the order in which objects are colored yellow?

They become yellow in increasing order of their “distance” from  $o$ .

4. (10) How do you know?

Let's try something that looks like induction on the distance from object  $o$ . We claim (and show below) that exactly all objects that are "distance"  $k$  from object  $o$  are colored yellow during the  $k$ th iteration of the outermost **Repeat** loop. The answer to the real question of interest then follows directly from this claim.

*The basis step:* Object  $o$  is the first to become yellow, it becomes yellow on iteration 0, it is distance 0 from object  $o$ , no other object is distance 0 from  $o$ , and no other object is colored yellow on iteration 0.

*The induction step:* On each iteration of the outermost **Repeat** loop existing brown colored objects become yellow and some blue objects become brown. Suppose for each iteration  $0, 1, \dots, k$  the objects becoming yellow are exactly those objects that are distance  $k$  away from  $o$ . Objects becoming yellow on iteration  $k + 1$  are those that became brown on iteration  $k$ . Objects becoming brown on iteration  $k$  are all one "hop" from those objects becoming yellow on iteration  $k$  due to the action of the middle and innermost **Repeat** loops on black box objects (which are all the ones made brown on the previous iteration), hence their distance to  $o$  cannot be greater than  $k + 1$ . Moreover, since their color changes from blue to brown (and they have not yet been colored yellow), they cannot be at a distance of less than  $k + 1$  from  $o$  (the induction hypothesis implies that all objects of distance  $k$  or less from  $o$  must be colored yellow up through iteration  $k$ ). Therefore all the objects becoming brown on iteration  $k + 1$  are exactly distance  $k + 1$  from  $o$  and on the next iteration all those objects will become yellow since they are all placed in the black box at the outset of iteration  $k + 1$ . This completes the proof of the inductive hypothesis.

5. (5) Will all the objects be colored yellow at the end of the algorithm?

Not always

6. (10) How do you know?.

By counter example. If the set of objects can be partitioned into two sets such that no object in one set connects with any object in the second set then the algorithm will stop with objects of one set colored yellow and objects of the other set colored blue.

7. (10) What is the complexity of the algorithm as a function of  $n$ ?

Since each pairing is considered at most one time and since there are usually many more pairings than objects, the complexity of the algorithm depends on the number of pairings which, in the worst case, is  $O(n^2)$ .

8. (5) We only need two colors for objects (the hardware store ran low on paint) if what data structure is used?

*Answer:* a Queue

9. (15) How would you use that data structure?

Instead of objects becoming brown, they will be put into a queue. The black box is now not needed because objects can be placed into the queue exactly at the moment they would have become brown. Objects will be pulled from the queue exactly at the moment they are to be colored yellow.

10. (10) To implement the red box and objects as classes in C++, state what variables you would need, the data structures you would use, and what methods you would implement to use them. Briefly describe how you would use the data structures.

An array of pointers to all objects. For each object, a linked list of pointers to the neighbors of the objects and a 'color' variable. The innermost **Repeat** loop walks through the list of  $x$ , removing nodes as they are considered (effectively removing a pairing from the red box), and coloring the right object brown if it is blue. There should also be a linked list of pointers to objects that grows as objects are colored brown. That list would be used to implement the black box.