Name: Key

Student #: ______________

CSE373 Summer 2016 Final
August 19th, 2016

Rules:

- The exam is closed-book, closed-note, closed-calculator, closed-electronic
- You have 60 minutes to complete the exam. **Please stop promptly at 11:50**
- If you write any answers on scratch paper, please clearly write your name on every sheet and write a note on the original sheet directing the grader to the scratch paper. **We are not responsible for lost scratch paper or for answers on scratch paper that are not seen by the grader due to poor marking**.
- Code/Pseudocode will be graded on proper behavior/output rather and not on style, unless otherwise noted.
- Unless otherwise stated, all **logs are base 2.**

Problem	Description	Earned	Max
1	Runtimes		10
2	Graphs		9
3	Critical thinking		12
4	Sorting		13
5	Parallelism & Concurrency		10
6	Maximum Spanning Tree		5
	Extra Credit		1
Total	Total Points		60

Advice:

- Read questions carefully. Understand a question before you start writing.
- Write down thoughts and intermediate steps so you can get partial credit. **Clearly circle your final answer**.
- The questions are not necessarily in order of difficulty. **Skip around**. Make sure you get to all of the questions.
- If you have questions, ask them.
- Any clarifications will be noted on the projector
- Take a deep breath, relax!

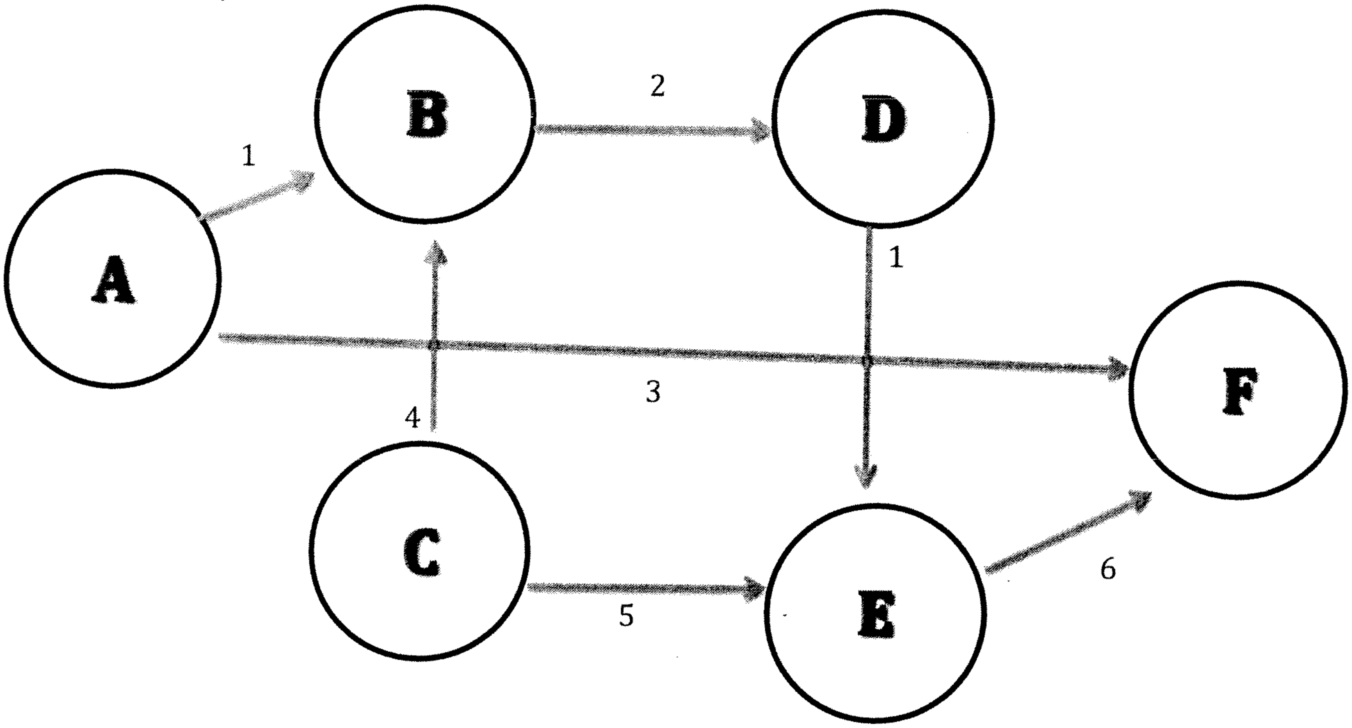
1) Runtimes (10 points)

Fill in this table with the tightest **worst-case** asymptotic running times of each operation when using the data structure listed. Assume the following:

- Items are comparable (given two items, one is less than, equal to, or greater than the other) in O(1) time.
- For insertions, it is the client's responsibility not to insert an item if there already is an equal item in the data structure (so the operations don't need to check this).
- For insertions, assume the data structure has enough room (do not include resizing costs).
- For deletions, assume we **do not** use lazy deletion.

	Insert	Contains	Delete	getMin	getMax
	(take an item and insert it into the data structure)	(take an item and return if it is in the structure or not)	(find an item and remove it from the structure, if present)	(return smallest element in the structure	(Return largest element in the structure)
Sorted Array	shift O(n)	O(log n)	shift O(n)	O(1)	O(1)
Unsorted Array	O(1)	O(n)	O(n)	O(n)	O(n)
Array MinHeap	O(log n)	O(n)	find first O(n)	O(1)	O(n)
AVL Tree	O(log n)				
Hashtable w/ Separate Chaining	O(1) prepend O(n) append	O(n)	O(n)	O(n)	O(n)

2) Graphs (9 points)



(6 points) For the above graph, do the following:

a) Provide a valid topological ordering of the nodes in the graph. If one doesn't exist, explain why.

ACBDEF or CABDEF

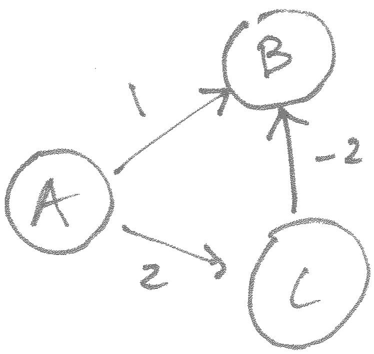
b) Circle all of the following that apply for the above graph (as depicted):

DIRECTED UNDIRECTED CYCLIC ACYLCIC

WEIGHTED UNWEIGHTED STRONGLY CONNECTED

WEAKLY CONNECTED DISCONNECTED

(3 points) Give an example where Dijkstra's algorithm gives the wrong answer in the presence of a negative-cost edge, but no negative-cost cycles. The example need not be complex, it is possible to demonstrate using as few as 3 vertices.



A→B: Dijkstra's says A→B, shortest is A→C→B.

3) Critical thinking (12 points)

Given two input arrays of integers and an output array, fill the output array with numbers that are present in both input arrays (if any).

For example:

array1: [5, 10, 3, 4, 1, 5], array2: [1, 8, 7, 3, 2, 4, 5], out: [ ]

myMethod(array1, array2, out);

array1: [5, 10, 3, 4, 1], array2: [1, 8, 7, 3, 2, 4, 5], out: [5, 1, 4, 3]

You will solve this question 2 times. First, optimize the minimization of runtime (**average case**). Second, optimize the minimization of auxiliary space. You may write java code or pseudocode. Your pseudocode should resemble real code structure and syntax. It is better to have an inefficient solution than no solution at all. **You may assume that no parameters are null, You may assume that neither of the two input arrays are empty. You may assume that the output array is initially empty, and that it is sufficiently large.**

Solution 1 (optimize for runtime):

```
myRuntimeOptimizedMethod(int[] arr1, int[] arr2, int[] output)
// your code here
int i=0;
Set<Integer> s= new HashSet<Integer>();
for(int x : arr1) {
   s.add(x);
}
for(int y: arr2) {
   if (s.contains(y)) {
      output[i] = y;
      s.remove(y)  // avoid duplicates in output
      i++;
   }
}
```

(space for solution 2 on next page)

One of many solutions.

Solution 2 (optimize for space):

```
mySpaceOptimizedMethod(int[] arr1, int[] arr2, int[] output)
// your code here
```

```
int i=0
for(int x: arr1) {
    for(int y: arr2) {
        if(x==y) {
            boolean duplicate = false
            for(int k: output) {
                if(x==k) {
                    duplicate = true;
                    break;
                }
            }
            if(!duplicate) {
                output[i] = x
                i++;
            }
        }
    }
}
```

4) Sorting (13 points)

(8 points) The following arrays are **partially sorted**, the result of a malicious TA interrupting the sorting algorithm being performed on each array. The algorithms were at least 1/4 finished when they were stopped.

Based on your knowledge of each sorting algorithm, determine which algorithm was being used to sort each array.

Choose from the following: *Insertion sort, selection sort, heap sort, merge sort, quick sort*

Lots of viable answers, we were lenient.

Array | Sort Used:

19	39	42	44	53	149	91	87	193	146

Sort used: selection

sorted "globally"

14	42	17	60	72	219	100	10	5	1

Sort used: heap

binary min heap   reverse sorted order

29	35	44	114	139	37	30	28	46	87

Sort used: insertion

sorted locally.

6	10	3	50	72	60	61	1	34	64

Sort used: merge

sorted partitions.

Sorting (continued)

(5 points) Answer the following questions, no explanations are needed.

a) What is the tightest worst case asymptotic runtime of quicksort, using a median of three, a linear time partition, that switches to insertion sort when there are less than 30 elements to sort?

$O(n^2)$

b) The best possible asymptotic runtime for comparison based sorting algorithms is $O(n)$. (TRUE / FALSE)

c) Heap sort can be done in-place. (TRUE / FALSE)

d) Merge sort can be done in-place. (TRUE / FALSE)

e) We can improve the asymptotic runtime of merge sort by switching to insertion sort when there are fewer than 500 elements to sort. (TRUE / FALSE)

5) Parallelism and Concurrency (10 points)

(3 points) Given a program where 75% of it is parallelizable (and 25% of it must be run sequentially), what is the maximum **speedup** you would expect to get with 3 processors? Note: **<u>you must show your work for any credit</u>.** For full credit give your final answer as a number or a simplified fraction (not a formula).

$$\text{speedup} = \frac{T_1}{T_p} = \frac{1}{S + \frac{(1-S)}{P}} = \frac{1}{.25 + \frac{.75}{3}} = \frac{1}{.5} = 2$$

(7 points) For each of the following, indidcate whether it can be easily computed with:

- A parallel map operation
- A parallel reduce operation
- Neither

No explanation is needed.

a) Given an array of strings, replace all strings of length < 5 with the String "h4cked"
(MAP / REDUCE / NEITHER)

b) Given an array of numbers, find the median
(MAP / REDUCE / NEITHER)

c) Given an array of strings, count the number of strings that are not the empty string ("")
(MAP / REDUCE / NEITHER)

For the following, circle true or false:

d) A computer with only 1 processor (1 core) can make use of parallelism (TRUE / FALSE)

e) A computer with only 1 processor (1 core) can make use of concurrency (TRUE / FALSE)

f) For a given block of code, the **span** is always less than the **work** (TRUE / FALSE)

g) For a given block of code, the **work** is always less than the **span** (TRUE / FALSE)

6) Maximum Spanning Tree (5 points)

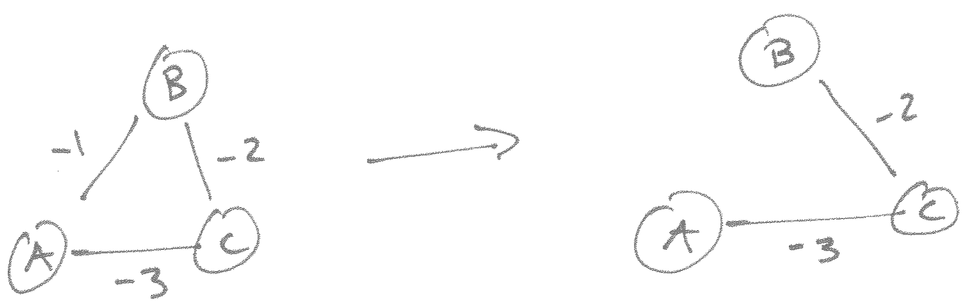
The Maximum Spanning Tree Problem is: given a connected, undirected graph with weights on its edges, find a spanning tree that maximizes the weights of its edges.

Explain how a Minimum Spanning Tree algorithm can be used to find the Maximum Spanning Tree of a graph. Credit will be awarded based on the efficiency of the algorithm you describe.

Be concise in your explanation.

1) negate the edge weights in your graph.

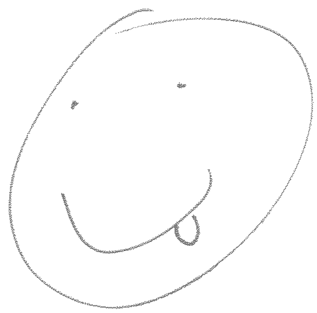
2) use MinST algorithm.



3) done! you may also negate the edges again, but this is a valid maximum spanning tree.

7) Extra credit (1 point):

Draw us a picture!



Blank sheet for extra space