Name: ______Solutions_____

UW ID #: ___________________________

CSE373 Spring 2015: Final

(closed book, closed notes, NO calculators allowed)

Instructions: Read the directions for each question carefully before answering. We may give partial credit based on the work you write down, so if time permits, show your work! Use only the data structures and algorithms we have discussed in class or that were mentioned in the book so far. The questions are not necessarily in order of difficulty, so skip around as you answer them.

Note: For questions where you are drawing pictures or showing work, **PLEASE CIRCLE YOUR FINAL ANSWER FOR ANY CREDIT.**

Good luck!

Total: 89 points. Time: 1 hour 50 minutes.

Question	Max Points	Score
1	7	
2	8	
3	8	
4	14	
5	6	
6	16	
7	17	
8	12	
9	1	
Total	89	

1) **Short Answer [7pt]**. Please give the best answer for each question.

(a) The idea that transistor density doubles roughly every 18 months is called ___Moore's Law___

(b) The idea that adding more processors has diminishing returns is called ____Amdahl's Law_____

(c) For an Array-based Stack with resizing, the worst case runtime of push is ________O(n)________

(d) For an Array-based Stack with resizing, the amortized runtime of push is ________O(1)________

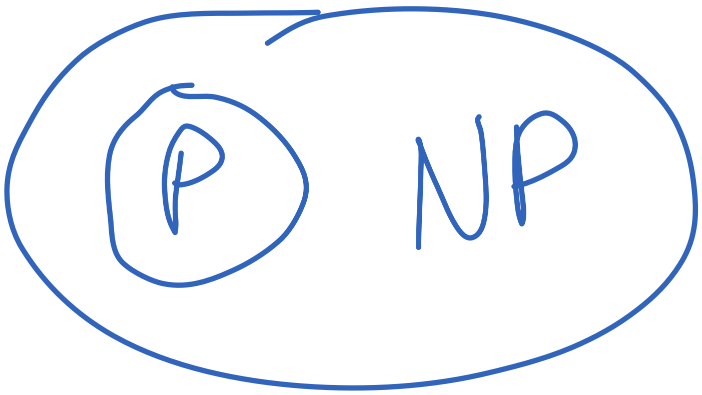
(e) Temporal locality is if an item (a location in memory) is referenced:

______________________________and it is soon after used again___________________________________

(f) Spatial locality is if an item (a location in memory) is referenced:

_________________________and items nearby in memory are used soon____________________________

(g) Draw the relationship between the sets P and NP based on what is currently known.



2) **Algorithms by Design [8pt]:** Label each algorithm by the most precise definition of its design. Possible choices are listed below.

Brute Force   Greedy   Divide & Conquer   Dynamic Programming   Backtracking

(a) Finding an exit of a maze on foot	_____Backtracking_____
(b) Merge Sort	__Divide & Conquer__
(c) Prim's	_______Greedy________
(d) Generating Fibonacci numbers	_Dynamic Programming_
(e) Cracking passwords	______Brute Force______
(f) Finding the shortest path	_Greedy and/or Dynamic Programming_
(g) Placing 8 Queens	_____Backtracking_____
(h) Parallel array summation	__Divide & Conquer__

3) **Preserving Abstractions [8pts]:** Suppose there is some class, Waldo, that has a constructor that takes an argument of type Foo and stores the content of the Foo object for later use by other methods. In the constructor for class Waldo, please write the code to store the Foo object with the minimum amount of copying needed to preserve the abstraction.

(a)

```
public class Bar {
      public int x;
      public int y;
      public Bar(int _x, int _y){ x = _x; y = _y; }
}
public class Foo {
      public Bar b;
      public Bar c;
      public Foo(Bar _b, Bar _c) { b = _b; c = _c; }
}
public class Waldo {
      public Foo a;
      public Waldo(Foo _a) {
            //your answer here:

            a = new Foo(new Bar(_a.b.x, _a.b.y), new Bar(_a.c.x,
      _a.c.y));
      }
}
```

(b)

```
public class Bar {
      public final int x;
      public final int y;
      public Bar(int _x, int _y){ x = _x; y = _y; }
}
public class Foo {
      public final Bar b;
      public final Bar c;
      public Foo(Bar _b, Bar _c) { b = _b; c = _c; }
}
public class Waldo {
      public Foo a;
      public Waldo(Foo _a) {
            //your answer here:

            a = _a;
      }
}
```

(c)
```
public class Bar {
      public int x;
      public int y;
      public Bar(int _x, int _y){ x = _x; y = _y; }
}
public class Foo {
      public final Bar b;
      public final Bar c;
      public Foo(Bar _b, Bar_c) { b = _b; c = _c; }
}
public class Waldo {
      public Foo a;
      public Waldo (Foo _a) {
            //your answer here:

            a = new Foo(new Bar(_a.b.x, _a.b.y), new Bar(_a.c.x,
      _a.c.y));
      }
}
```

(d)
```
public class Bar {
      public final int x;
      public int y;
      public Bar(int _x, int _y){ x = _x; y = _y; }
}

public class Foo {
      public final Bar b;
      public Bar c;
      public Foo(Bar _b, Bar _c) { b = _b; c = _c; }
}
public class Waldo {
      public Foo a;
      public Waldo(Foo _a) {
            //your answer here:

            a = new Foo(new Bar(_a.b.x, _a.b.y), new Bar(_a.c.x,
      _a.c.y));
      }
}
```

4) **Parallelizing Quick Sort [14pts]**: Below we have provided you with the sequential code for quicksort.

```
// Sorts the section of the array
public void quickSort(int[] arr, int lo, int hi){
      if(lo >= hi) {
            return;
       }
      int pivotIndex = partition(arr, lo, hi);
      quickSort(arr, lo, pivotIndex - 1);
      quickSort(arr, pivotIndex + 1, hi);
}

// Uses arr[lo] as the pivot and returns the location of the pivot
public int partition(int[] arr, int lo, int hi) { … }

public void sort(int[] arr){
      quickSort(arr, 0, arr.length);
}
```

(a) Transform the sequential code into parallel code using the skeleton code provided below:

```
public class QSThread extends java.lang.Thread {
      int lo, hi;
      int[] arr;

      public QSThread(int[] a, int l, int h) {
            arr = a;
            lo = l;
            hi = h;
      }

      public void run() {

            // Base Case
            if(lo >= hi) {
                  return;
            }
            int pivotIndex = partition(arr, lo, hi);

            // Generate subproblems
            QSThread left = new QSThread(arr, lo, pivotIndex - 1);
            QSThread right= new QSThread(arr, pivotIndex + 1, hi);

            // Spawn off one thread, handle the other
            left.start();
            right.run();
            left.join();




      }

      public int partition(int[] arr, int lo, int hi) { … }
}

public void sort(int[] arr) {
      QSThread qs = new QSThread(arr, 0, arr.length);
      qs.run();
}
```

(b) What is the work done by the parallelized version of quicksort in the worst case?

$O(n^2)$

(c) What is the work done by the parallelized version of quicksort in both the best case?

$O(n \log n)$

(d) What is the span of the parallelized version of quicksort in the worst case?

$O(n^2)$

(e) What is the span of the parallelized version of quicksort in the best case?

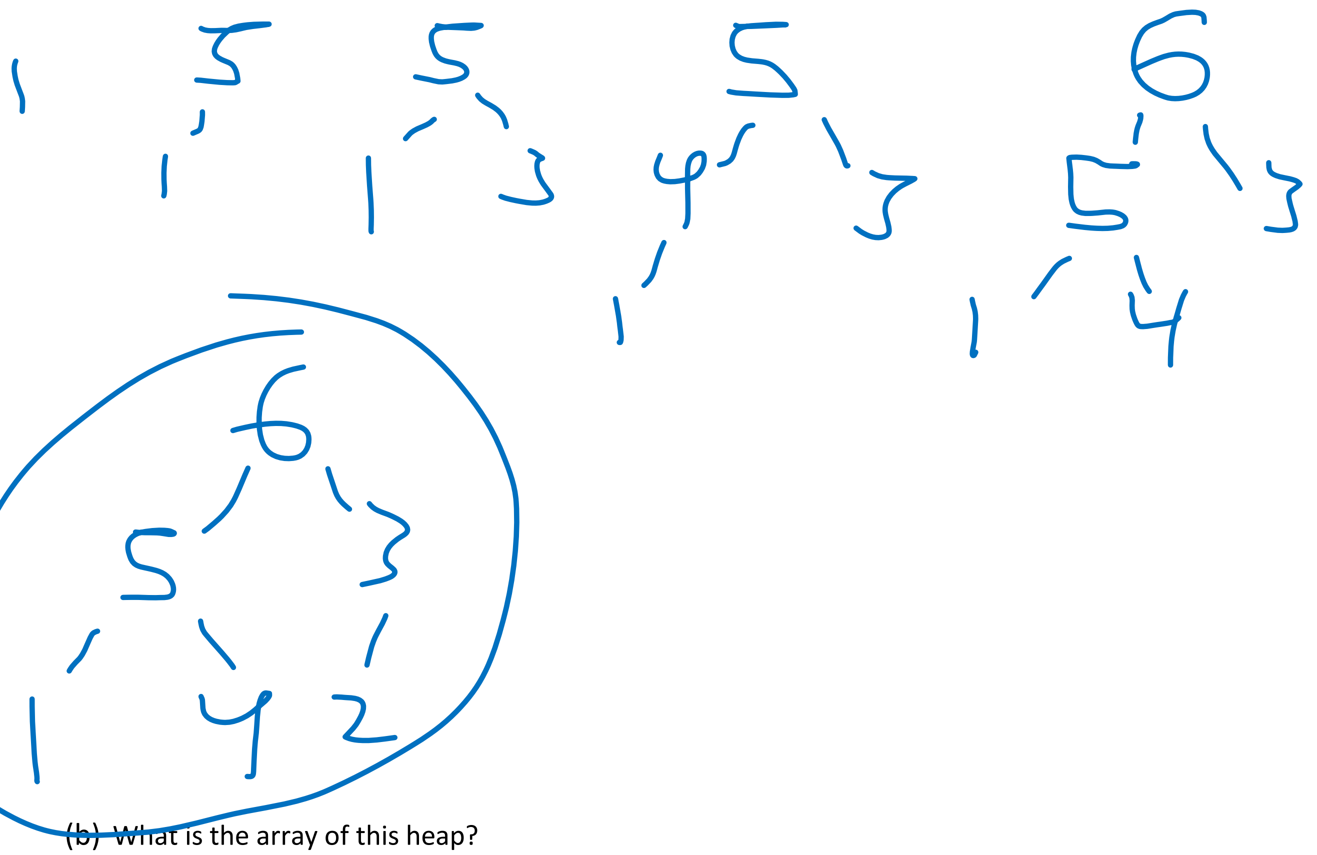
$O(n)$

5) **Hash Table [6pts]:** Write the code to add lazy deletion to a Hash Table full of integers greater than 0. The table presumes that values that equal 0 are unfilled entries. Values are inserted using linear probing. We will presume this table does not resize. Presume that we have provided the rest of the code, fill out the delete method with lazy deletion. You can add fields or helper methods if needed. Please indicate with a comment in the code how you chose to represent a deleted item.

```
Class HashTable() {
      int[] arr;
      int size;


      HashTable() {
            arr = new int[127];
            size = 0;
      }

      void insert(int x) {…}
      boolean find(int x) {…}
      int hash(int x) {…}

      void delete(int x) {

            int index = hash(x) % arr.length;

            while(arr[index] != 0) {
                  if(arr[index] == x) {
                        //Use negative numbers for lazy deletion
                        arr[index] = -1;
                        size--;
                        return;
                  }
                  index++;
                  index %= arr.length;
            }




      }
}
```

6) **Sorting [16pts]**: Sort the array [1 5 3 4 6 2] using **HeapSort**, using a **Max**Heap and sorting in-place.

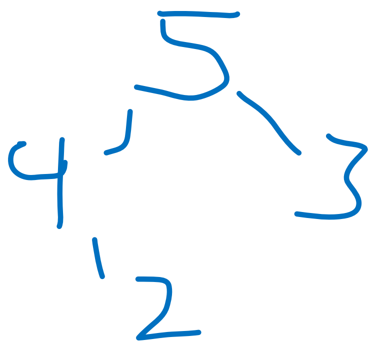
(a) Build the heap using insertions. Show your work and circle the final tree.



(b) What is the array of this heap?

6	5	3	1	4	2

(c) Start sorting the array by doing the first delete**Max** operation. Draw the tree at the end of this step and the partially sorted array after this step.



$$\underbrace{5 \quad 4 \quad 3 \quad 1 \quad 2}_{\text{heap}} \quad \underbrace{6}_{\text{sorted array}}$$

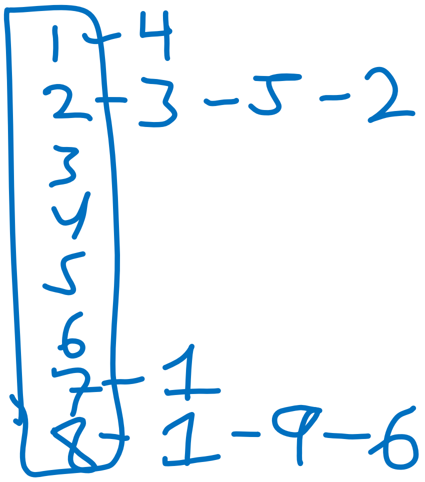
Sorting, continued:

(d) Sort the array [3 1 4 1 5 9 2 6] using MergeSort, draw the recursion tree showing how it splits and merges.

```
                3 1 4 1 5 9 2 6

        3 1 4 1                 5 9 2 6

    3 1         4 1         5 9         2 6

    1 3         1 4         5 9         2 6

        1 1 3 4                 2 5 6 9

                1 1 2 3 4 5 6 9
```

(e) Sort the following <key, value> pairs using **stable** Bucket Sort:

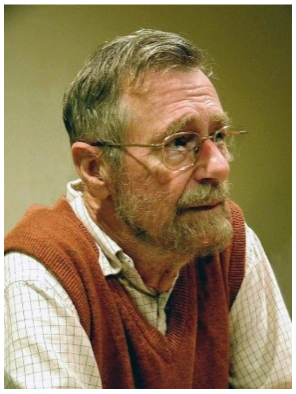
<2, 3>   <7, 1>   <1, 4>   <8, 1>   <2, 5>   <8, 9>   <2, 2>   <8, 6>



<1, 4>   <2, 3>   <2, 5>   <2, 2>   <7, 1>   <8, 1>   <8, 9>   <8, 6>

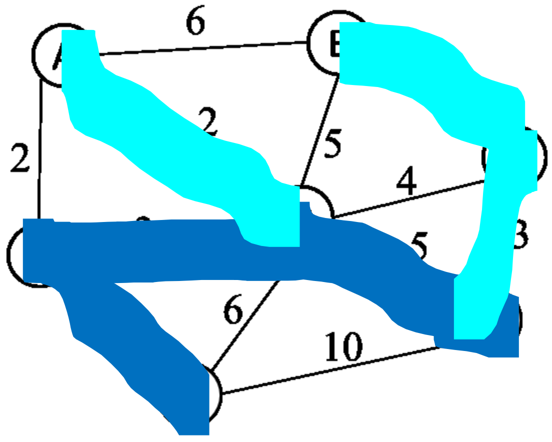
7) **Graph Algorithms [17pts]**:

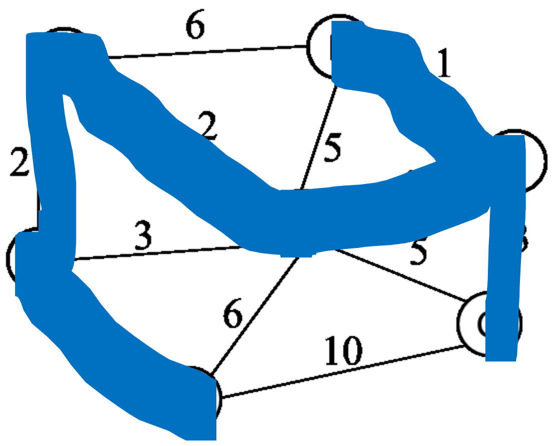
a) Dijkstra's and Prim's algorithms are very similar. Circle the edges selected by Dijkstra's algorithm finding the shortest path from G to F in the left graph. Circle the edges selected by Prim's algorithm on the right.



Dijkstra's Edges

Prim's Edges





Cyan edges are optional, showing the edges

that Dijkstra selected to other nodes

Graph Algorithms, continued:

(b) What edit could you make to Dijkstra's algorithm to make it Prim's algorithm?

Change what the cost is for determining what vertex to add to the known cloud. In Dijkstra's algorithm, the cost is the shortest path thus far from the start vertex. In Prim's algorithm, the cost is the cost to add the vertex to the known cloud.

(c) Will Prim's algorithm and Kruskal's algorithm always return the same tree? Explain.

Not always. There can be multiple minimum spanning trees. In that case, since the two algorithms select the edges in different orders, they may end up with different trees.

While implementing Dijkstra's algorithm, you can use a priority queue to keep track of the next-closest Vertex to add. When visiting a node you have already seen, you can either 1) Decrease the key of that Vertex entry or 2) Add redundant vertices and while popping out the next closest Vertex ignore entries of Vertices that you have already seen before. For each question, answers in terms of |V| and |E| using tight asymptotic bounds. Presume that |E| > |V|.

(d) Using **decreaseKey()**, what is 1) The time it takes to insert a Vertex into the queue and 2) The space can the queue take up?

Time: ___O(log|V|)__ Space: ___O(|V|)___

(e) Using **redundant vertices**, what is 1) The time it takes to insert a Vertex into the queue and 2) The space can the queue take up?

Time: _ O(log|V|) or O(log|E|)_ Space: ___ O(|E|)__

(f) Using the priority queue with decreaseKey(), what is the runtime of Dijkstra's algorithm?

___ O(|E|log|V|)___

8) **Selecting the best data structure [12pts]**: For the following scenarios, indicate the best data structure to use. Make sure to be specific.

(a) You want to be able to add, find, remove, and produce a sorted list of students which are stored by their ID number relatively quickly using the least amount of memory.

AVL Tree

(b) You want to store student profiles for fast retrieval using their student ID as a key.

HashTable

(c) A company has sold out of a product and wants to keep track of which orders to fill when the product comes back in stock.

Queue

(d) You have a bunch of family trees and given two people you want to see if they have a common ancestor.

UpTree / Union Find

(e) You want to create a social networking site and want to track the friendships between people all over the world.

Adjacency List

(f) You want to keep track of friendships in a closely knit group.

Adjacency Matrix

9) **Bonus [1pt]:** What is your favorite data structure?

Any answer.

SCRATCH PAPER:

SCRATCH PAPER: