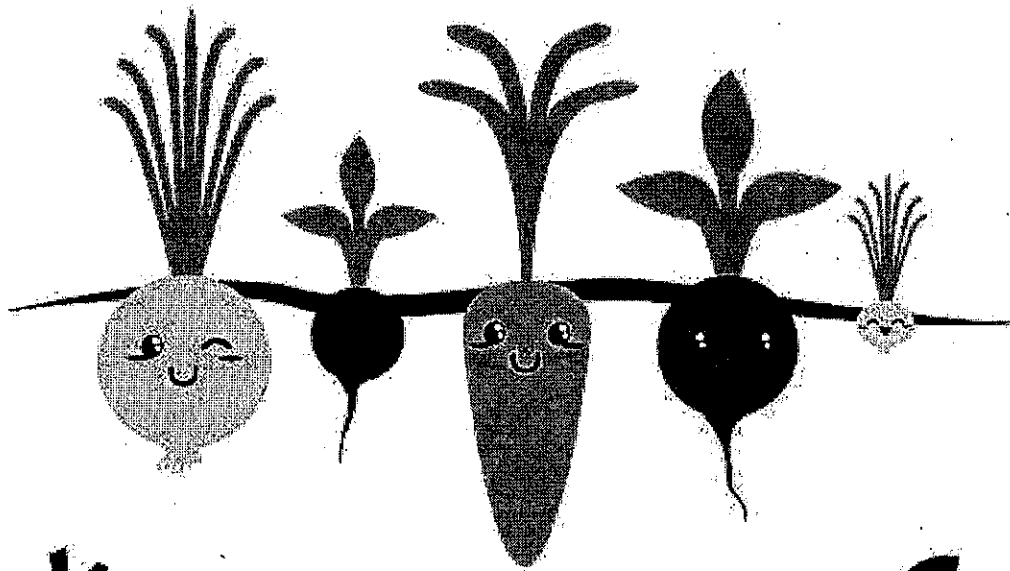


# *Chris and Richard's Final Prep Packet*

*A Premium Selection of Handcrafted, Artisanal Practice Problems*



**WE'RE ROOTING  
FOR YOU!**

Hashtables - 16 Au	2
Sorting (Simple Algorithms) - 15 Wi	3
Sorting (Complex Algorithms) - 15 Wi	4
Graphs (Path-finding) - 12 Wi	5
Graphs (MST) - 12 Su	6
Parallelism (ForkJoin) - 15 Wi	7
Parallelism (Theory) - 15 Wi	8
Parallelism (Parallel Prefix) - <del>15 Wi</del> 16 Au	8
Concurrency - 12 Wi	9
P, NP - 15 Wi	11

## Hashtables - 16 Au

### 1) [12 points total] Hash Tables

For a) and b) below, insert the following elements in this order: 50, 21, 29, 10, 39, 19. For each table, TableSize = 10, and you should use the primary hash function  $h(k) = k \% 10$ . If an item cannot be inserted into the table, please indicate this and continue inserting the remaining values.

a) Quadratic probing hash table

0	50
1	21
2	
3	39
4	10
5	
6	
7	
8	19
9	29

b) Separate chaining hash table – use a linked list for each bucket where the values are ordered by increasing value

0	→ 10 → 50
1	→ 21
2	
3	
4	
5	
6	
7	
8	
9	→ 19 → 29 → 39

c) What is the load factor in Table b)?

0.6

d) In a sentence or two, describe double hashing.

The first hash determines the original location to place. If there is a collision, then the second hash function is used to determine the probing step distance.

e) What is one advantage of double hashing over quadratic probing, be specific.

Double hashing avoids secondary clustering. Double hashing avoids the issue of not finding a viable bucket when the table is half full.

f) What is the big-O worst case runtime of a find operation on a table like table b)?

$O(n)$

g) What is the big-O worst case runtime of an Insert in a separate chaining hash table containing  $N$  elements where each bucket points to an AVL tree?

$O(\log n)$

Sorting (Simple Algorithms) - 15 Wi

1) [10 points total] Sorting: (Assume that array `sun[]` has indices: 0 to `size-1`)

```
SunnySort(int[] sun) {  
    for (int i = 1; i < size; i++) {  
        int j;  
        int temp = sun[i];  
        for (j = i; j > 0 && temp < sun[j-1]; j--) {  
            sun[j] = sun[j-1];  
        }  
        sun[j] = temp;  
    }  
}
```

a) [2 points] This is actually a sort mentioned in class. What sort is this?

Insertion Sort

b) [4 points] Describe the best and worst case data set for this sort. (If all cases behave similarly, please state that.) What is the big-O running time of those two data sets?

Best Case data set?

Sorted Order

Best Case running time?

$O(n)$

Worst Case data set?

Reverse Sorted Order

Worst Case running time?

$O(n^2)$

c) [2 points] Is it an in-place sort? Why or why not?  
(no credit without a reason or a definition of in-place, for partial credit define in-place sorting)

Only uses constant extra space, Yes

d) [2 points] Is it a stable sort? Why or why not?  
(no credit without a reason or definition of stable, for partial credit define stable sorting)

Tied elements preserve order, Yes

## Sorting (Complex Algorithms) - 15 Wi

### 8) [12 points] Sorting

- a) **Radix Sort:** Give a formula for the worst case big-O running time of radix sort. For full credit, your formula should include all of these variables:

max\_value – the values to be sorted range from 0 to max\_value  
 radix – the radix or base to be used in the sort  
 n – the number of values to be sorted.

Answer:

$$O(\log_{\text{radix}}^{\text{max\_value}} \cdot (n + \text{radix}))$$

- b) **Quicksort:** Give the recurrence for each of the following: (Note: We are NOT asking for the closed form.)

Quicksort (parallel sort & parallel partition) – best case span

Answer:

$$T(n) = O(\log n) + T\left(\frac{n}{2}\right)$$

Quicksort (parallel sort & parallel partition) – worst case span

Answer:

$$T(n) = O(\log n) + T(n-1)$$

- c) Give big-O runtimes of the following in terms of n. For parallel sorts, give the span.

$O(n \log n)$  Mergesort (sequential) – worst case

$O(\log^2 n)$  Quicksort (parallel sort & parallel partition) – best case span

$O(n \log n)$  Quicksort (parallel sort & parallel partition) – worst case span

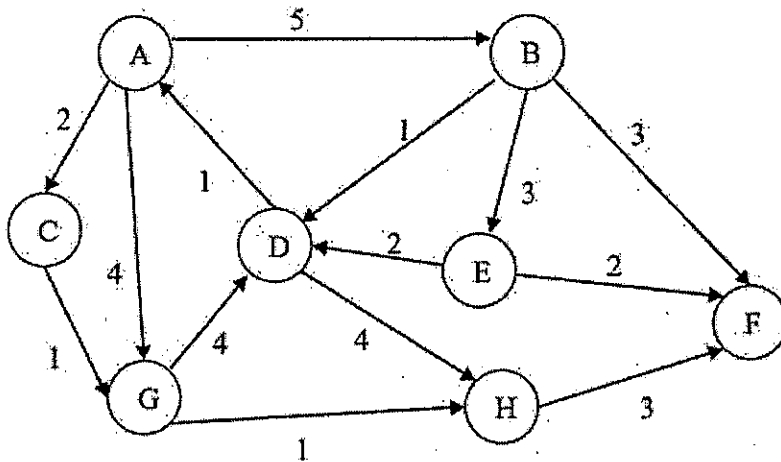
$O(n \log n)$  Quicksort (sequential) – best case

$O(n^2)$  Quicksort (sequential) – worst case

Graphs (Path-finding) - 12 Wi

4) 10 points : Single-Source Shortest Paths

Consider the following directed, weighted graph:



- a) Step through Dijkstra's algorithm to calculate the single-source shortest paths from vertex *A* to every other vertex. Show your steps in the table below. Cross out old values and write in new ones, from left to right in each cell, as the algorithm proceeds. Also list the vertices in the order which Dijkstra's algorithm marks them known:

Order vertices marked as known: A C G H B D F E

Vertex	Known	Distance	Path
A	X	0	-
B	X	5	A
C	X	2	A
D	X	7 <del>6</del>	<del>6</del> B
E	X	8	B
F	X	7	H
G	X	4 <del>3</del>	A C
H	X	4	G

- b) What is the lowest-cost path from *A* to *F* in the graph, as computed above?

A → C → G → H → F

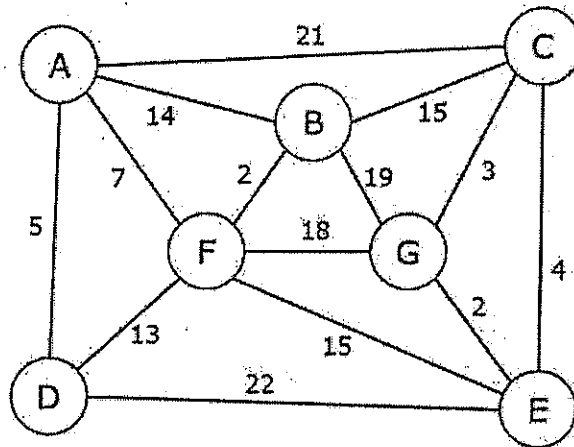
- c) To guarantee correctness of Dijkstra's algorithm, all edge costs must be non-negative. Imagine the edge from *A* to *B* had cost -3. Why would this make it impossible for any algorithm to provide a correct answer for single-source shortest paths?

Negative Cycle A → B → D → A

Graphs (MST) - 12 Su

6. (continued)

For your convenience, here is the graph again.



- (c) Step through Kruskal's algorithm to calculate a minimum spanning tree of the graph. Show your steps in the table below, including the disjoint sets at each iteration. If you can select two edges with the same weight, select the edge that would come alphabetically last (e.g., select E—F before B—C). Also, select A—F before A—B).

Edge Added	Edge Cost	Running Cost	Disjoint Sets
-	-	0	(A) (B) (C) (D) (E) (F) (G)
E-G	2	2	(A) (B) (C) (D) (E G) (F)
B-F	2	4	(A) (B F) (C) (D) (E G)
C-G	3	7	(A) (B F) (C) (D) (C E G)
A-D	5	12	(A D) (B F) (C E G)
A-F	7	19	(A B D F) (C E G)
E-F	15	34	(A B C D E F G)

## Parallelism (ForkJoin) - 15 Wi

4) [10 points] In Java using the ForkJoin Framework, write code to solve the following problem:

- **Input:** An array of ints (does not contain duplicates)
  - **Output:** the maximum value and its location (index) in the input array.
- Do not employ a sequential cut-off: the base case should process one element. (You can assume the input array will contain at least one element.)
  - Give a class definition, FindMax, along with any other code or classes needed.
  - We have provided some of the code for you, you should also fill in the \_\_\_\_\_ part.

```
import java.util.concurrent.ForkJoinPool;
import java.util.concurrent.RecursiveTask;
```

```
class Pair { // You are not required to use this class
    int a;
    int b;
    public Pair (int a, int b) {
        this.a = a;
        this.b = b;
    }
}
```

```
class Main{
    static final ForkJoinPool fjPool = new ForkJoinPool();
    Pair findMax (int[] array) {
        return fjPool.invoke(new FindMax( 0, array.length, array ));
    }
}
```

```
public class FindMax extends RecursiveTask<Pair> {
    int low, high;
    int[] array;
    public FindMax (int low, int high, int[] array) {
        ...
    }
    protected Pair compute() {
        if (high - low < 2) {
            return new Pair (low, array[low]);
        } else {
            int mid = (low + (high - low) / 2);
            FindMax left = new FindMax (low, mid, array);
            FindMax right = new FindMax (mid, high, array);
            left.fork();
            Pair rResult = right.compute();
            Pair lResult = left.join();
            return left (lResult.b) > rResult.b ? rResult lResult : rResult;
        }
    }
}
```



Ahmed's Law  $\frac{T_1}{T_p} = \frac{1}{S + (\frac{1-S}{P})} = \frac{1}{0.25 + (\frac{0.75}{5})} = \frac{1}{0.25 + 0.15} = \frac{1}{0.4} = \frac{5}{2} = 2.5$

Parallelism (Theory) - 15 Wi

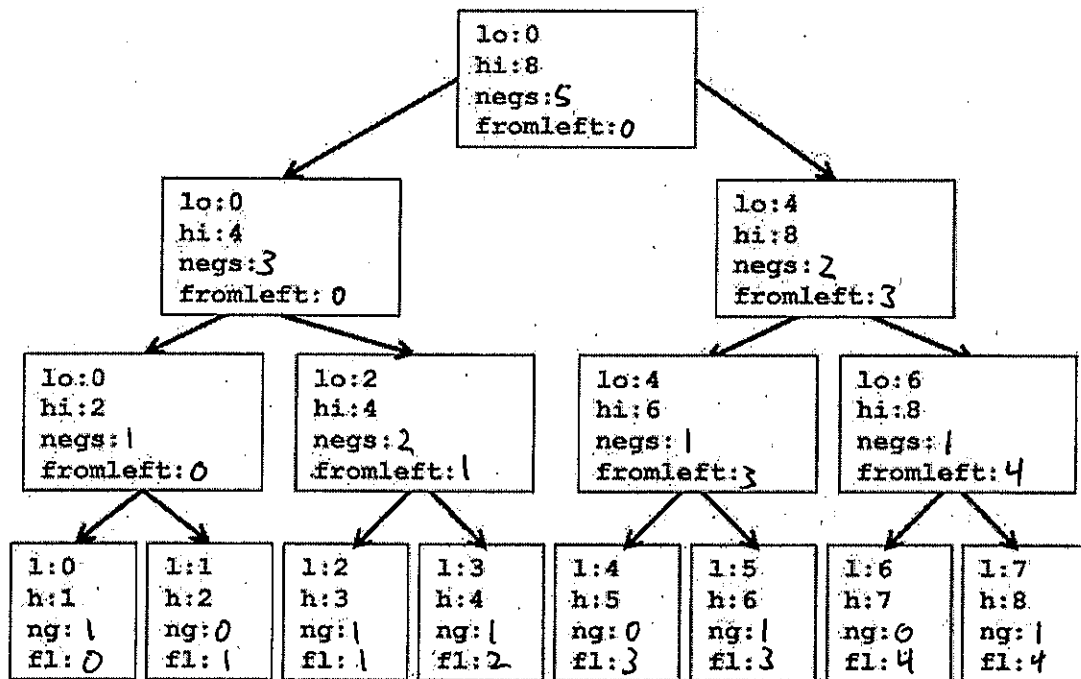
5) [6 points] Speedup

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 5 processors. Note: You must show your work for any credit. For full credit give your answer as a number or a simplified fraction (not a formula).

Parallelism (Parallel Prefix) - ~~15 Wi~~ 16 Au

4) [10 points] Parallel Prefix CountNegatives:

a) Given the following array as input, perform the parallel prefix algorithm to fill the output array with the number of negative values contained in all of the cells to the left (including the value contained in that cell) in the input array. Fill in the values for *negs*, and *fromLeft* in the tree below. Do not use a sequential cutoff. Note: This is NOT sum!



Index	0	1	2	3	4	5	6	7
Input	-3	5	-4	-7	2	-8	9	-5
Output	1	1	2	3	3	4	4	5

b) How is the *fromLeft* value computed for a node in the tree? Specifically, if you have a node with *negs* & *fromLeft* computed, how do you compute *fromLeft* for its left & right children (both of which have *negs* already computed).

Left child's *fromLeft*:

Parent's *fromLeft*

Right child's *fromLeft*:

Parent's *fromLeft* + sibling's *neg*

## Concurrency - 12 WI

### 7) 10 points : MoveToFrontList Concurrency

Consider this pseudocode for a MoveToFrontList, which is correct in a sequential context. It does not map keys to data items, but instead just tests whether a key is in the list.

```
01: class Node {
02:   Key key;
03:   Node next;
04:
05:   Node(...) { // Constructor that stores these 2 fields }
06: }
07:
08: class MoveToFrontList {
09:   Node front = null;
10:
11:   void insert(Key key) {
12:     front = new Node(key, front);
13:   }
14:
15:   Boolean contains(Key find) {
16:     if(front == null) { return false; }
17:     if(front.key == find) { return true; }
18:
19:     Node prev = front;
20:     Node current = front.next;
21:     while(current != null) {
22:       if(current.key == find) {
23:         prev.next = current.next;
24:         current.next = front;
25:         front = current;
26:         return true;
27:       }
28:       current = current.next;
29:     }
30:     return false;
31:   }
32: }
```

We have numbered the lines of code so that you can reference them in your answers. Please do this, as it will be faster and will keep your answer more concise.

- a) Describe an interleaving of insert("a") and contains("b") that results in the insert("a") being "missed" (i.e., contains("a") will return false).

B: 6-24

A: 11-13

D: 25-31

Line 25 overwrites insert

- b) Describe an interleaving of insert("a") and insert("b") that results in the insert("a") being "missed" (i.e., contains("a") will return false).

B:  $11 - 12\frac{1}{2}$  after constructor but before assignment

A:  $11 - 13$

B:  $12\frac{1}{2} - 13$

- c) Describe an interleaving of contains("a") and contains("a") that results in them returning different values (i.e., one returns true and one returns false).

A1: 15-23

A2 misses A as it is being moved

A2: 15-31

A1: 24-31

- d) Using any of the mutual exclusion mechanisms discussed in lecture (including those unique to Java), describe how to fix this class so that it is correct in concurrent usage. Your only concern is correctness (i.e., performance is not a concern).

Synchronize both methods

P, NP - 15 Wi

10) [8 points] P, NP, NP-Complete

a) "NP" stands for Nondeterministic Polynomial

b) What does it mean for a problem to be in NP?

Given a proposed solution, we can verify whether the solution is correct in polynomial time

c) Give two examples of NP-Complete problems:

Hamiltonian Circuit and Traveling Salesman

d) What should you do if you determine the problem you are trying to solve is NP-complete, yet you still need to solve it?

Use an approximation algorithm that guarantees something close or a heuristic that seems to work well most of the time. The problem may also be restricted by a pattern and you can find a optimal solution for that case.