

## Homework Week 4

### Text Questions

Monica Quaintance  
mjq2102@columbia.edu

#### 1. Weiss, Exercise 7.1. Show the intermediate steps.

*Sort the sequence 3, 1, 4, 1, 5, 9, 2, 6, 5 using insertion sort.*

```
3| 1 4 1 5 9 2 6 5
1 3| 4 1 5 9 2 6 5
1 3 4| 1 5 9 2 6 5
1 1 3 4| 5 9 2 6 5
1 1 3 4 5| 9 2 6 5
1 1 3 4 5 9| 2 6 5
1 1 2 3 4 5 9| 6 5
1 1 2 3 4 5 6 9| 5
1 1 2 3 4 5 5 6 9|
```

#### 2. Weiss, Exercise 7.19.

*Sort 3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5 using quicksort with median-of-three partitioning and a cutoff of 3.*

Top level sort (recursion 0 level):

Mof3(3,9,5) = 5, 5 == pivot.

Sort 1st, mid, and last, and partition:

```
[3 1 4 1 5 3 2 6 5 |5 9]
```

```
[3 1 4 1 (5) 3 2 6 (5) |5 9]
           i           j
```

```
[3 1 4 1 5 3 2 6 5 |5 9]
           j i
```

```
[3 1 4 1 5 3 2 5 5 6 9]
```

```
small: [3 1 4 1 5 3 2 5]
```

```
large: [5 6 9]
```

Sort small side (recursion 1 level):

Mof3(3,1,5) = 3, 3 == pivot.

Sort 1st, mid, and last, and partition:

[1 1 4 3 5 3 2 5]

[1 1 4 2 5 3 |3 5]

[1 1 (4) 2 5 (3) |3 5]  
          i          j

[1 1 3 2 (5) 4 |3 5]  
          j  i

[1 1 3 2 3 4 5 5 ]

small: [1 1 3 2]

large: [3 4 5 5]

Sort small of small (recursion 2 level):

Mof3(1,1,2) = 1, 1 == pivot.

Sort 1st, mid, and last, and partition:

[1 3 |1 2]

[1 3 |1 2]  
     j i

[1 1 3 2]

small: [1 1]

large: [3 2]

insertion sort returns: [1 1 2 3]

Sort large of small (recursion 2 level):

Mof3(3,4,5) = 4, 4 == pivot.

Sort 1st, mid, and last, and partition:

[3 5 |4 5]

j i

insertion sort returns: [3 4 5 5]

Insertion sort takes care of large of top (recursion 1 level):

insertion sort returns: [5 6 9]

Final array:

[1 1 2 3 3 4 5 5 5 6 9]

### 3. Weiss, Exercise 7.28a.

*When implementing quicksort, if the array contains lots of duplicates, it may be better to perform a three-way partition (into elements less than, equal to, and greater than the pivot), to make smaller recursive calls. Assume three-way comparisons, as provided by the `compareTo` method.*

*a. Give an algorithm that performs a three-way in-place partition of an  $N$ -element subarray using only  $N - 1$  three-way comparisons. If there are  $d$  items equal to the pivot, you may use  $d$  additional Comparable swaps, above and beyond the two-way partitioning algorithm. (Hint: As  $i$  and  $j$  move toward each other, maintain five groups of elements as shown below):*

EQUAL   SMALL   UNKNOWN   LARGE   EQUAL  
          i                   j

```
int i = left, j = right - 1;
int leftPivots = 0;
int rightPivots = 0;

for( ; ; ){

    while( a[ ++i ].compareTo( pivot ) < 0 ){ }
    while( a[ --j ].compareTo( pivot ) > 0 ){ }

    if (a[j].compareTo(pivot) == 0){
        swapReferences(a, a[j], left + 1); //swap small side pivots to left
        leftPivots++;
    }

    if (a[i].compareTo(pivot) == 0){
        swapReferences(a, a[i], right - 2); //swap large side pivots to right
    }
}
```

```

        rightPivots++;
    }

    if(i<j)
        swapReferences( a, i, j );
    else
        break;
}

// return right elements + pivot
for(int k = 0; k < rightPivots + 1; k++){
    swapReferences( a, i + k, right - (k+1) );
}

// return left elements
for(int k = 0; k < leftPivots; k++){
    swapReferences( a, j - k, left + (k+1) );
}

```

#### 4. Weiss, Exercise 9.1.

Find a topological ordering for the graph in Figure 9.81.

A	2	1	1	0	0	-	-	-	-	-	-
B	1	1	1	1	1	0	-	-	-	-	-
C	3	3	3	3	3	3	2	1	1	0	-
s	0	-	-	-	-	-	-	-	-	-	-
D	2	1	0	-	-	-	-	-	-	-	-
E	4	4	3	2	1	0	0	-	-	-	-
F	2	2	2	2	2	2	2	1	0	-	-
t	3	3	3	3	3	3	3	3	2	1	0
G	1	0	-	-	-	-	-	-	-	-	-
H	1	1	0	0	-	-	-	-	-	-	-
I	2	1	1	1	1	1	1	0	-	-	-

Enq: s G D, H A - B, E - I F C t  
 Deq: s G D H A B E I F C t

Ordering: [s,G,D,H,A,B,E,I,F,C,t]

#### 5. Weiss, Exercise 9.10a.

Explain how to modify Dijkstra's algorithm to produce a count of the number of different minimum paths from  $v$  to  $w$ .

Dijkstra's algorithm keeps track of the length of the shortest path to a given vertex. You could keep track of another value as well, the number of shortest paths that are equal to that length.

When you know  $v$ 's shortest path, you can mark it as known. You have kept track of the number of shortest paths to  $v$ . Then, for every adjacent vertex  $w$  to  $v$  you can add the number of shortest paths to  $w$  (through  $v$  or not) to the count of shortest paths to  $v$ .

If the shortest path to  $w$  through  $v$  is less than the length of the existing path, then replace the count of  $w$  with the count of  $v$ . If it's the same length, then add the count of  $v$  to the count of  $w$ .

## 6. Weiss, Exercise 9.15.

*a. Find a minimum spanning tree for the graph in Figure 9.84 using both Prim's and Kruskal's algorithms.*

Vertices:

(A,B) 3 Accepted

(B,C) 10 Rejected

(D,A) 4 Accepted

(A,E) 4 Rejected

(E,B) 2 Accepted

(B,F) 3 Accepted

(F,C) 6 Rejected

(C,G) 1 Accepted

(D,E) 5 Rejected

(E,F) 11 Rejected

(F,G) 2 Accepted

(D,H) 6 Rejected

(H,E) 2 Accepted

(E,I) 1 Accepted

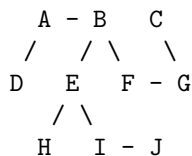
(I,F) 3 Rejected

(F,J) 11 Rejected

(J,G) 8 Rejected

(H,I) 4 Rejected

(I,J) 7 Accepted



b. Is this minimum spanning tree unique? Why?

No, not unique, you can connect the vertices in more than one configuration.

## 7. Weiss, Exercise 9.38a and 9.38b.

You are given a set of  $N$  sticks, which are lying on top of each other in some configuration. Each stick is specified by its two endpoints; each endpoint is an ordered triple giving its  $x$ ,  $y$ , and  $z$  coordinates; no stick is vertical. A stick may be picked up only if there is no stick on top of it.

a. Explain how to write a routine that takes two sticks  $a$  and  $b$  and reports whether  $a$  is above, below, or unrelated to  $b$ . (This has nothing to do with graph theory.)

Compute the ranges of the two sticks on the  $x$  and  $y$  axes:

$$R(a_{start}(x), a_{end}(x)) = R_{ax} \quad R(a_{start}(y), a_{end}(y)) = R_{ay}$$

$$R(b_{start}(x), b_{end}(x)) = R_{bx} \quad R(b_{start}(y), b_{end}(y)) = R_{by}$$

If either of:

- 1) the intersection of the  $x$ -ranges of  $a$  and  $b$
- 2) the intersection of the  $y$ -ranges of  $a$  and  $b$

are zero, then the two sticks are not in the same place.

If the intersection of both ranges  $\neq 0$ , then calculate the point at which the two sticks cross (intersection of the two lines in the  $x$ - $y$  planes) and the stick with the higher  $z$ -value at that point is on top.

b. Give an algorithm that determines whether it is possible to pick up all the sticks, and if so, provides a sequence of stick pickups that accomplishes this.

Can do this with Kruskal's algorithm. Every stick is a vertex in its own disjoint set. The graph is "directed" in that a stick on top of another outranks the one below it.

If stick  $A$  is on the top of the stack (all things below it), then  $A$  is the set of picked up sticks. If stick  $B$  is below  $A$ , then add  $B$  to the set of  $A$ . If a stick has nothing on top of it, it can be added to the set of  $A$ , repeat until no sticks remain (essentially topological sort it). If there is no cycle, you can pick up all the sticks, otherwise not.