CS317: Algorithms Homework Assignment #4

Due: See Canvas for Assignment Due Dates (20 points)

NOTE: NO LATE ASSIGNMENTS WILL BE ACCEPTED because we often review them in class on the due date.

Please upload the document containing your answers. They can be handwritten and scanned, but they must be clearly legible to receive a grade on the assignment. PDF is the best format for canvas. You DO NOT Need to include this cover sheet in your upload. It is formatted for the grader to use for me, as needed.

Chapter 4: Work the following problems. Point values are provided for each problem.

Problem #	Points	Grader's Notes
Sec 4.1, #11	*4*	
Sec 4.2, #1	*2*	
Sec 4.2, #5	*1*	
Sec 4.4, #3a,b	*2*	*Updated – took out c,d parts*
Sec 4.4, #4	*1*	
Sec 4.4, #5	*1*	

Chapter 5: Work the following problems. Point values are provided for each problem.

Problem #	Points	Grader's Notes
Sec 5.1, #1 a-d	*4*	
Sec 5.1, #5b	*1*	
Sec 5.1, #8a	*3*	
Sec 5.1, #7	*1*	

Other ungraded practice problems:

Sec 4.1: #12

Sec 4.4: #10 (fake coin problem)

4.1 # 11 Nolan Anderson

11. Let $A[0..n-1]$ be an array of n sortable elements. (For simplicity, you may assume that all the elements are distinct.) A pair $(A[i], A[j])$ is called an ***inversion*** if $i < j$ and $A[i] > A[j]$.

a. What arrays of size n have the largest number of inversions and what is this number? Answer the same questions for the smallest number of inversions.

Inversion if $1 < 3$ and $A[1] > A[3]$

$9{,}999 < 10{,}000$ and $A[9{,}999] > A[10{,}000]$ (high → low)

Looking for an array where each subsequent element is smaller than its previous element. Array of size $n = \infty$, number of inversions is $n-1$

$[(n-1), (n-2) \ldots 1+0] = \frac{n(n-1)}{2}$

Not an inversion if $i > j$ and $A[i] < A[j]$ — Looking for an array that is sorted low to high.

$[0+1 \ldots (n+1)+(n+2)] \hookrightarrow \frac{n(n+1)}{2}$

or $i < j$ and $A[i] < A[j]$ — $n = \infty$, inversions $= 0$.

or $i > j$ and $A[i] > A[j]$

$A = 1, 2, 3, 4, 5, \ldots$ (indices $0\ 1\ 2\ 3\ 4 \ldots$)

$A[0] < A[1], A[1] < A[2] \ldots$

b. Show that the average-case number of key comparisons in insertion sort is given by the formula

$$C_{avg}(n) \approx \frac{n^2}{4}.$$

$$C_{worst}(n) = \sum_{i=1}^{n-1} \sum_{j=0}^{i-1} 1 = \sum_{i=1}^{n-1} i = \frac{(n-1)n}{2} \in \Theta(n^2).$$

$$C_{best}(n) = \sum_{i=1}^{n-1} 1 = n - 1 \in \Theta(n).$$

$I = \sum X_{[i,j]}$

$E[I] = E[\sum x_{ij}] = \sum E[x_{ij}]$

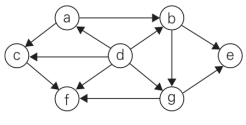
$E[x_{ij}] = PR[x_{ij} = 1] = PR[A_i \text{ and } A_j]$

$E[I] = \sum E[x_{ij}] = \sum (1/2)$

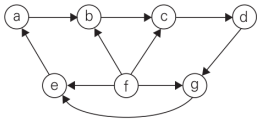
$E[I] = \frac{n(n-1)}{4}$

4.2 #1 #5

1. Apply the DFS-based algorithm to solve the topological sorting problem for the following digraphs:

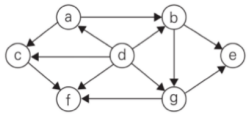


(a)



(b)

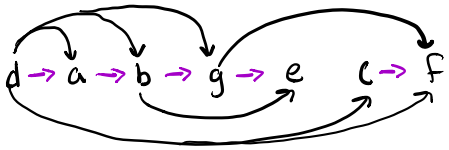
a.)



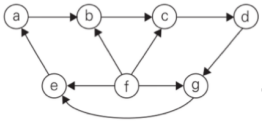
(a)

Popping off order: e, f, g, b, c, a, d

Topologically sorted list



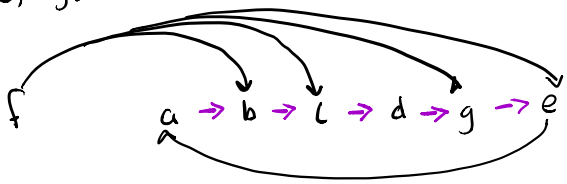
b.)



(b)

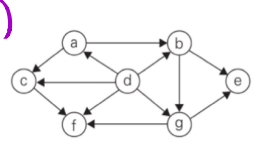
popping off order

e, g, d, c, b, a, f



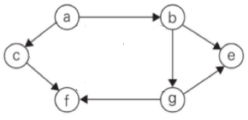
5. Apply the source-removal algorithm to the digraphs of Problem 1 above. delete source w/ no incoming edges

a.)



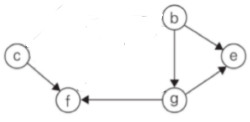
(a)

delete d ...



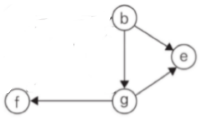
(a)

delete a ...



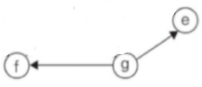
(a)

delete c



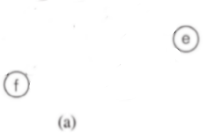
(a)

delete b ...



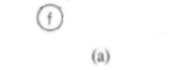
(a)

delete g



(a)

delete e...

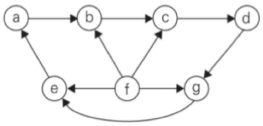


(a)

delete f: Solution obtained is:

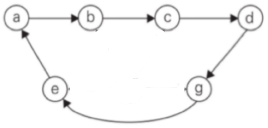
d, a, c, b, g, e

b)



(b)

delete f



(b)

Not solvable, no more sources.

4.4 #3 a, b #4 #5

3. a. What is the largest number of key comparisons made by binary search in searching for a key in the following array?

0												12
3	14	27	31	39	42	55	70	74	81	85	93	98

13 elements

$C_{worst}(n) = C_{worst}(\lfloor n/2 \rfloor) + 1$

$C_{worst}(13) = C_{worst}(6) + 1$ (3)

$C_{worst}(6) = C_{worst}(3) + 1 = 2$ (1)

$C_{worst}(3) = C_{worst}(1)$

Largest number of comparisons = 3

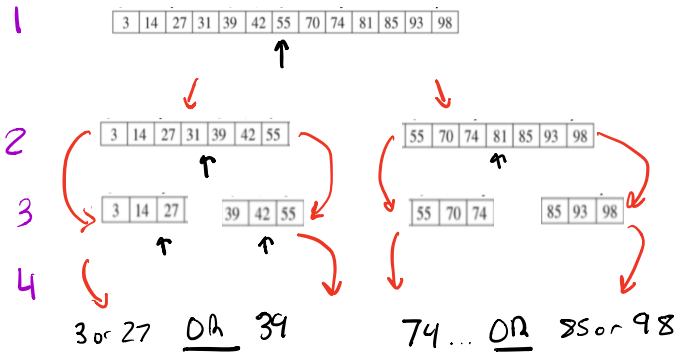
$C_{worst}(n) = \log_2(n+1)$

$= \log_2(14)$

Largest # of key comparisons = 4

b. List all the keys of this array that will require the largest number of key comparisons when searched for by binary search.

There are a lot of different ways...



4. Estimate how many times faster an average successful search will be in a sorted array of one million elements if it is done by binary search versus sequential search.

$n = 1,000,000$

Binary search: $\log_2(n) = 19.93$ comp.

Sequential search: $\frac{(n+1)}{2} = \frac{1,000,001}{2} =$

500,000.5 comparisons.

Binary search is 25089.83 times faster

5. The time efficiency of sequential search does not depend on whether a list is implemented as an array or as a linked list. Is it also true for searching a sorted list by binary search?

I would say it does not hold true for binary search because you would need to get to the element. This would add more if statements and adds/subs, which would make it take longer

5.1 #1 a-d #5b #7 #8a

1. a. Write pseudocode for a divide-and-conquer algorithm for finding the position of the largest element in an array of n numbers.

```
Largest element (S, e)
S= Starting position; e= end position
if s=e return 1
else
    mid= (s+e)/2
    position 1 = largest element (Array[mid])
    position 2 = largest element (Array[mid+1..e])
    if Array[position 1] ≥ Array[position 2]
        return position 1
    else return position 2
```

b. What will be your algorithm's output for arrays with several elements of the largest value?

It will output the left most largest element.

c. Set up and solve a recurrence relation for the number of key comparisons made by your algorithm.

$$C(n) = C\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + C\left(\left\lceil \frac{n}{2} \right\rceil\right) + 1, \quad n > 1, \quad C(1) = 0$$

$$\rightarrow C(2^k) = C\left(\left[\frac{2^k}{2} \right]\right) + C\left(\left[\frac{2^k}{2} \right]\right) + 1$$

$$= C\left(\left[2^{k-1}\right]\right) + C\left(\left[2^{k-1}\right]\right) + 1$$

$$C(2^k) = 2C\left(\left[2^{k-1}\right]\right) + 1$$

$$= 2C\left(2\left[2^{k-1}\right]\right) + 1 + 1$$

$$= 2^2 C\left(2\left[2^{k-1}\right] + 1\right) + 2 + 1$$

$$\cdots$$

$$2^i C\left(2^{k-i}\right) + 2^{i-1} + 2^{i-2} + \cdots + 1$$

$$C(2^k) = 2^k C\left(2^{k-k}\right) + 2^{k-1} + 2^{k-2} + \cdots + 1$$

$$= 2^k C(1) + \cdots + 1$$

$$= 2^k (0) + 2^{k-1} + 2^{k-2} + \cdots + 1$$

$$= 2^{k-1} + 2^{k-2} + \cdots + 1$$

$$= 2^k - 1$$

$$\boxed{C(n) = n - 1}$$

$C(n) = 2C\left(\frac{n}{2}\right)+1$

$= 2C\left(\frac{n-1}{2}-1\right)+1$

$= 2\left(\frac{n-2}{2}\right)+1$

$= n-1$ ✓

d. How does this algorithm compare with the brute-force algorithm for this problem?

Same number of comparisons, but my solution has recursion overhead.

5. Find the order of growth for solutions of the following recurrences.

b. $T(n) = 4T(n/2) + n^2,\ T(1) = 1$

$O(n^2)$ -> quadratic

7. Is mergesort a stable sorting algorithm?

Yes

8. a. Solve the recurrence relation for the number of key comparisons made by mergesort in the worst case. You may assume that $n = 2^k$.

$C_{worst}(n) = 2C_{worst}(n/2) + n - 1$

$C(2^k) = 2\left(\frac{2^k}{2}\right) + 2^k - 1$

$= 2\left(\frac{2^{k-1}}{2}\right) + 2^k - 1 = 2^2[2C_{worst}(2^{k-2}) - 1]$

$= 2^2 C_{worst}(2^{k-2}) + 2 \cdot 2^k - 2 - 1$

$= 2^k C_{worst}(2^{k-k}) + k2^k - 2^{i-2} - \ldots - 1$

$= k2^k - (2^k - 1)$

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