CSCE 221 Cover Page

Homework Assignment #2

First Name: Mitesh Last Name: Patel UIN: 124002210

User Name: patel221881 E-mail address: patel2218812@tamu.edu

Please list all sources in the table below including web pages which you used to solve or implement the current homework. If you fail to cite sources you can get a lower number of points or even zero, read more on Aggie Honor System Office website: `http://aggiehonor.tamu.edu/`

Type of sources				
People	TA	Peer Teacher		
Web pages (provide URL)	Listed Below			
Printed material	Data Structures and Algorithms Text Book			
Other Sources				

Webpages: http://www.algolist.net/Algorithms/Sorting/Quicksort
https://en.wikipedia.org/wiki/Merge_sort
www.piazza.com
http://www.cplusplus.com/reference/stack/stack/
http://www.cplusplus.com/reference/queue/queue/
http://www.cplusplus.com/forum/beginner/69889/

I certify that I have listed all the sources that I used to develop the solutions/codes in the submitted work.
On my honor as an Aggie, I have neither given nor received any unauthorized help on this academic work.

Your Name Mitesh Patel Date 10-17-15

Homework 2

due October 18 at 11:59 pm.

1. (20 points) Linked list questions.

   (a) Write a recursive function in C++ that counts the number of nodes in a singly linked list.

   i. int count (*ListNode L){
   ii. if (L == NULL) return 0;
   iii. else return 1+ count (L->next);
   iv. }

   (b) Write a recurrence relation that represents the running time for your algorithm.

   i. T(n) = T(n-1) + 1
   ii. T(0) = 1

   (c) Solve this relation and provide the classification of the algorithm using the Big-O asymptotic notation.

   i.

$T(0)=1$

$T(n)=T(n-1)+1$

$T(n-1)=T(n-2)+1$

$T(n-2)=T(n-3)+1$

$T(n-3)=T(n-4)+1$

$$\begin{aligned}
T(n) &= T(n-1)+1 \\
&= T(n-2)+1+1 = T(n-2)+2 \\
&= T(n-3)+1+1+1 = T(n-3)+3 \\
&= T(n-4)+1+1+1+1 = T(n-4)+4 \\
&\vdots \\
&= T(n-k)+k
\end{aligned}$$

$k_{max}=n$

$$\begin{aligned}
&= T(0)+n \\
&= n+1 = \boxed{O(n)}
\end{aligned}$$

2. (20 points) Write a recursive function that finds the maximum value in an array of int values without using any loop

(a)

```
int findmax(int A[], int size, int max)
{
    int i = size - 1; // set index 1 less than size every time recursive function executes
    int maxval = max; // get the max passed and store in variable
    if(i==0)
    {
        if(A[i]>A[i+1]) maxval = A[i]; // check first two elements in array
     return maxval; // if no more elements to check then return the maxval
    }

    if(maxval < A[i] || maxval < A[i-1]) // as long as no other element is greater than max va
    {
        if (A[i] > A[i-1]) maxval = A[i]; // set max val
        findmax(A,i,maxval); // call function again to keep checking the array

    }

    return findmax(A, i, maxval);

}
```

(b) Write a recurrence relation that represents running time of your algorithm.

i. T(n) = T(n-1) + 1

ii. T(0) = 0

(c) Solve this relation and classify the algorithm using the Big-O asymptotic notation.

i.

$T(0) = 0$

$T(n) = T(n-1) + 1$

$T(n-1) = T(n-2) + 1$

$T(n-2) = T(n-3) + 1$

$T(n-3) = T(n-4) + 1$

$T(n) = T(n-1) + 1$

$= T(n-2) + 1 + 1 = T(n-2) + 2$

$= T(n-3) + 1 + 1 + 1 = T(n-3) + 3$

$= T(n-4) + 1 + 1 + 1 + 1 = T(n-4) + 4$

$\vdots$

$= T(n-k) + K$

$k_{max} = n$

$= T(0) + n$

$= n = \boxed{O(n)}$

3. (10 points) What data structure is the most suitable to determine if a string is a palindrome? A string is a palindrome if it is equal to its reverse. For example, “racecar” and “so many dynamos” are palindromes (spaces are removed from many word strings). Justify your answer. Use Big-O notation to classify the running time of your algorithm.

   (a) A data structure that is most suitable to determine if a string is a palindrome would be a STL /ADT queue. This is because we can pop each string element and push it in a queue, and then we can pop element from the queue and push it onto a string. Consequently if it is a palindrome then the new string should be same as the orignal string. We can implement this using two for loops iterating through the string to pop and the queue to push into another string, therefore our runtime for this algorithm would be O(n).

4. (10 points) Solve C-5.2 on p. 224

   (a) We can use the queue Q to pop each element from S and enqueue it in Q, and then dequeue element from Q and push it onto S. This will reverse S. Then we repeat this, but look for element x by passing elements through Q and to S again so we can get the originial order back.

5. (20 points) What is the amortized cost of the stack push operation when the additional stack-array memory is allocated by each of these two strategies? Do calculations to support your answer.

(a) Doubling strategy – double the size of the stack-array memory if more memory is needed.

i. Doubling Strategy 0

- Assume initial capacity of array is 1. If this array is full we double its capacity. So we get 1, 2, 4, 8, 16 ...

Let c_i = cost of i-th push operation.

$c_i = i$ if array is full, that is we need to include the cost of additional $i-1$ copy operations (from smaller array to the larger one) plus one push operation.

To insert n objects, we replace array $k = \log_2 n$ times

So: $c_i \begin{cases} i & \text{if } i-1 \text{ is exact power of } 2 \\ 1 & \text{otherwise} \end{cases}$ 0

Total cost:

$$\sum_{i=1}^{n} c_i \leq n + \sum_{j=0}^{\log_2 n} 2^j < n + 2n = 3n$$

Since at most n operations that cost 1

Amortized cost : $\frac{3n}{n} = 3 = \boxed{O(1)}$

(b) Incremental strategy – increase the size of the stack-array by a positive constant c if more memory is needed.

i. Incremental

increase size by constant c if no more space

To insert n objects, we replace array $k = \frac{n}{c}$ times

$T(n)$ - total time of a series of n insert operations

$T(n)$ is proportional to $n + c + 2c + 3c + 4c + \ldots + kc$

$n + c(1 + 2 + 3 + \ldots + k) = n + ck(k+1)/2$

$T(n)$ is $O(n + k^2)$

amortized time is $\boxed{O(n)}$

6. (10 points) Describe (in pseudo code) how to implement the stack ADT using two queues. What is the running time of the push and pop functions in this implementation?

   (a) Psuedo code

      i. while(!Q1.isEmpty()) then do
      ii. Q2.push(Q1.pop())
      iii. end while
      iv. while(!Q2.isEmpty()) then do
      v. Q1.push(Q2.pop())
      vi. end while

   (b) Running time and psuedo code description

      i. We can enqueue elements into Q1 whenver a push is executed. This would be O(1). However, our pop takes O(n) because we are dequeing from Q2 and enqueing into Q1. Whatever we pop we enqueue into Q2, and then we pop from Q2 and enqueue to Q1 so we can get a stack like feature.

7. (10 points) Solve C-5.8 on p. 224

   (a) We can evaluate a expression that is in postix form using tokens. Lets say, 123+*45 -/ we can get tokens from the postfix expression. If the token is a operator then we push the value associated with it onto the stack. If it is a binary operator we pop two values from the stack, apply the operator to it, and push the result back into the stack. If unary operator then pop one value and push result back into the stack.

8. (20 points) Consider the quick sort algorithm.

(a) Provide an example of the inputs and the values of the pivot point for the best, worst and average cases for the quick sort.

i. Worst case:

A. Input: 8,7,6,5,4,3,2,1

B. Starting pivot points: 8 or 1.

ii. Best case:

A. Input: 8,7,6,5,4,3,2,1

B. Starting pivot point (median): 5, then median of the divided lists and so on

iii. Average case:

A. Input: 8,7,6,5,4,3,2,1

B. Starting pivot point can be any number that is not the best or worse case pivots, so: 7,6,5,4,3 or 2.

(b) Write a recursive relation for running time function and its solution for each case.

i.

Worst Case $T(1) = 0$

Recursive Relaton: $T(n) = T(n-1) + n$

$T(n-1) = T(n-2) + n - 1$

$T(n-2) = T(n-3) + n - 2$

$T(n-3) = T(n-4) + n - 3$

$= T(n-2) + n - 1 + n$

$= T(n-3) + n - 2 + n - 1 + n$

$= T(n-4) + \underbrace{n - 3 + n - 2 + n - 1 + n}$

→ We recognize this summation as $\frac{n(n-1)}{2}$

Therefore, worst case: $O(n^2)$

ii. Best case:

A. $T(n) = 2T(\frac{n}{2}) + n$, $T(1) = 0$

B. Using master theorm we see that a = 2, b=2, d=1, b^d= 2, so a = b^d and therefore, O(nlogn).

iii.

Average

Recurrence Relation $\begin{cases} T(n) = T(\alpha n) + T((1-\alpha) n) + n \\ T(1) = 0 \end{cases}$ $\alpha < 0.5$

assume: $\alpha = 1/3$ $\quad T(n) = T(\frac{n}{3}) + T(\frac{2}{3}n) + n$

Drawing tree:

left $1/3$
right $2/3$

$H_L < H_R$

n — n

$\frac{n}{3}$ $\quad \frac{2}{3}n$ — n

$\frac{n}{9}$ $\quad \frac{2}{9}n$ $\quad \frac{2}{9}n$ $\quad \frac{4}{9}n$ — n

H_L $\quad H_R$

$H_L = \log_3 n$ $\qquad H_R = \log_{3/2} n$

$O(H_R \cdot n)$

$= O(n \cdot \log_{3/2} n)$ — $\log_{3/2} n = \log_2 n \cdot \frac{1}{\log_2 3/2}$

$\boxed{= \Theta(n \log_2 n)}$

9. (15 points) Consider the merge sort algorithm.

(a) Write a recurrence relation for running time function for the merge sort.

i. $T(n) = 2T(\frac{n}{2}) + n$

ii. $T(1) = 0$

(b) Use two methods to solve the recurrence relation

i.

Iterative Method:

$T(n) = 2T(\frac{n}{2}) + n$

$T(1) = 0$

$T(\frac{n}{2}) = 2T(\frac{n}{4}) + n/2$

$T(\frac{n}{4}) = 2T(\frac{n}{8}) + n/4$

$T(\frac{n}{8}) = 2T(\frac{n}{16}) + n/8$

$$= 2(2T(\tfrac{n}{4}) + \tfrac{n}{2}) + n = 2^2 T(\tfrac{n}{4}) + 2n$$
$$= 4(2T(\tfrac{n}{8}) + \tfrac{n}{4}) + 2n = 2^3 T(\tfrac{n}{8}) + 3n$$
$$= 8(2T(\tfrac{n}{16}) + \tfrac{n}{8}) + 3n = 2^4 T(\tfrac{n}{16}) + 4n$$
$$\vdots$$
$$= 2^k T(\tfrac{n}{2^k}) + kn$$

$K_{max} = \log_2 n$

$$= 2^{\log_2 n} T\left(\frac{n}{2^{\log_2 n}}\right) + \log_2 n \cdot n$$
$$= 2^{\log_2 n} \cdot T(1) + \log_2 n \cdot n$$
$$= n \cdot \log_2 n = \boxed{O(n\log_2 n)}$$

Master theorem:

$$f(n) = \begin{cases} O(n^d) & a < b^d \\ O(n^d \log_b n) & a = b^d \\ O(n^{\log_b a}) & a > b^d \end{cases}$$

$f(n) = af(\frac{n}{b}) + cn^d$

$T(n) = 2T(\frac{n}{2}) + n$

$a = 2$, $b = 2$, $d = 1$; $b^d = 2$; $a = 2$; $a = b^d$

So using master theorem: $\boxed{O(n \log_2 n)}$

(c) What is the best, worst and average running time of the merge sort algorithm? Justify your answer.

i. The best, worse, and average cases for the merge sort are all O(nlogn). This is because it is a divide and conquer algorithm. For worst case, at every level of the recursive tree we end up doing O(n) work, and since it splits in half at each level the height would be logn. So we do this logn times, therefore O(nlogn), and since merge sort does not depend on order of data the best and average is the same as the worst case.