Istanbul Bilgi University

CMPE 211 Data Structure and Algorithms

2017-2018 Fall Midterm Exam Answers

Name	:	Department	: ....
Student No	:	Date	: ....
		Grade	: ....

Answers to the midterm questions.

[20P] Q.1 (a) What is the time complexity of the following program. (b) Propose a modification in the code, in order to reduce the time complexity. Then calculate the time complexity of your proposal.

```
public long power(int x, int n){
    if (n == 0) return 1;
    if (n%2 == 0)
        return power(x,n/2) * power(x,n/2);
    else
        return x * power(x,n/2) * power(x,n/2);
}
```

S.1 (a) $T(n) = 1 + 2T(n/2)$ with $T(1) = 1$. By induction, we obtain, $T(n) = n-1+nT(1) = 2n-1 = O(n)$. (b) Redundant code: $power(x, n/2)$ is called twice. Better agorithm can be

```
public long power(int x, int n){
    if (n == 0) return 1;
    long p =  power(x,n/2);
    if (n%2 == 0) return p*p;
    else return x * p*p;
}
```

Now, we have $T(n) = 1+T(n/2)$ with $T(1) = 1$ which is $O(log_2 n)$.

[20P] Q.2 Compare the running times for two algorithms T_A and T_B running on different computers A and B, over input size $n = 10^7$. What is your conclusion?

	Computer Power
A	10^{10} instructions per sec.
B	10^7 instructions per sec.

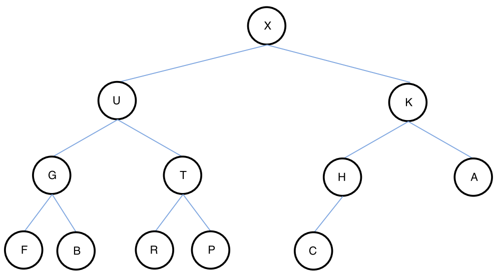
	Algorithm Time
A	$T_A(n) = n^2$
B	$T_B(n) = n + 2T_B(n/2)$ with base case: $T_B(1) = 1$

S.2 Lets calculate $T_B(n)$ first. $T_B(n) = n + 2T_B(n/2) = 2n+2^2T_B(n/2^2) = \ldots = kn+2^kT_B(n/2^k)$. If we assume $n = 2^k$ and $T_B(1) = 1$, we have $T_B(n) = n + nlog_2 n$. Now,

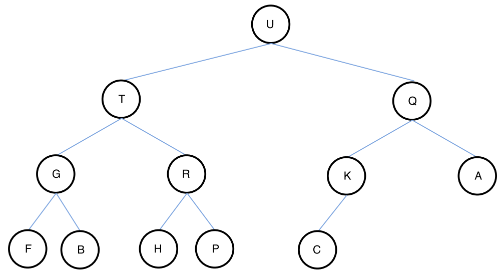
- A requires $n^2 = 10^7 \times 10^7 = 10^{14}$ instructions and it can do 10^{10} instructions per sec. Time required for A is $\frac{10^{14}}{10^{10}} = 10000$ sec.
- B requires $n + nlog_2 n = 10^7 + 10^7 log_2(10^7)$ instructions. So, we divide it to its speed and get $\frac{10^7+10^7 log_2(10^7)}{10^7} = 1 + log_2(10^7) = 24 sec$

Take home message: better algorithms can beat supercomputers.

[20P] Q.3 Max-Heap. (a) Give the array representation of the heap. (b) Insert item Q to the binary heap. Indicate any entries that changed. (c) Remove max and show resulting array and tree.



S.3 Results
(a) - X U K G T H A F B R P C Q
(b) - X U Q G T K A F B R P C H
(c) - U T Q G R K A F B H P C -



[20P] Q.4 Suppose you have implemented memory of an agent as an ordered-array. Memory holds information about a set of items. Indicate the worst-case running time of each operations below.

know(item)	*does item exist in the set, if so return its index.*	
learn(item)	*add unknown item to its correct place*	
forget(item)	*delete the given item from the set*	
recommend()	return the item who has max value in the set	
rank(item)	return the number of items in the set that are less than given item	

S.4 Since array is sorted, we can use binary search for **know(item)** and **rank(item)** in $O(log_2(n))$ time. Adding and deleting an item, requires exchanges to keep resulting array ordered. So in the worst case, **forget(item)** and **learn(item)** can be done in $O(n)$ time. **recommend()** is a constant time operation $O(1)$.

know(item)	$O(log_2(n))$
learn(item)	$O(n)$
forget(item)	$O(n)$
recommend()	$O(1)$
rank(item)	$O(log_2(n))$

[20P] Q.5 (a) Describe how merge sort operates? (b) What is its main disadvantage compared to quick sort? (c) Write the array content after all intermediate *merging* steps during the merge-sort.

Original Array	9	2	8	7	1
First Merge					
Second Merge					
Third Merge					
Fourth Merge					
Fifth Merge					

S.5 (a) it divides the array into two halves a[lo..mid] and a[mid+1 .. hi]. Then it recursively sorts the first half then the second half. Finally two halves are merged. (b) Additional memory. (c)

Original Array	9	2	8	7	1
First Merge	2	9	8	7	1
Second Merge	2	8	9	7	1
Third Merge	2	8	9	1	7
Fourth Merge	1	2	7	8	9
Fifth Merge	-	-	-	-	-

Alphabetical Order A-B-C-D-E-F-G-H-I-J-K-L-M-N-O-P-Q-R-S-T-U-V-W-X-Y-Z

Logarithm $log_2(10^6) = 19$ and $log_2(10^7) = 23$