Branden Vennes
CS 324
Homework 1
1/23/2019

PROBLEM 1

Express the function $n^3/(1000-100n^2-100n+3)$ in terms of Θ-notation.

n^3
/
$(1000-100n^2-100n+3)$

>> $\Theta(n^3)$

PROBLEM 2

Use mathematical induction to show that when n is an exact power of 2, the solution of the following recurrence is T(n) = n lg n:

T(n) =
{
2 if n=2;
2T(n/2)+n if n=2^k, for k>1
}

Base Case: n = 2
When n is 2, T(n) = 2. 2 * lg(2) = 2 * 1 = 2. Base case proved.

Induction Hypothesis: For k-1, T(n)=n*lg(n) when $n=2^{k-1}$.
Let k be a positive integer. Let $n=2^k$.
Then T(n)=2*T(n/2)+n.
$n/2=2^k/2=2^{k-1}$
So $T(n)=2*T(2^{k-1})+2^k$.
By the Induction Hypothesis, $T(n)=2*[(2^{k-1})*lg(2^{k-1})]+2^k$
$T(n) = 2*[2^{k-1}*(k-1)]+2^k$
$T(n) = 2^k*(k-1)+2^k$
$T(n) = 2^k[(k-1)+1] = 2^k * k$
$n * lg n = 2^k * lg(2^k) = 2^k * k$
Therefore, T(n) = n * lg n.

By the method of mathematical induction. The solution to the recurrence is T(n) = n lg n when n is an exact power of 2.

PROBLEM 3

Describe a Θ(n lg n)-time algorithm that, given a set S of n integers and another integer x, determines whether or not there exist two elements in S whose sum is exactly x.

1. Sort set S using merge sort.

2. Let value left be the first index of S and right be the last index of S.

```
left = 0
right = S.length - 1
```

3. If left == right, return false.

4. Take the sum of left and right.

5. If the sum is larger than x, then decrement right and go back to step 3, otherwise go to step 6.

6. If the sum is less than x, then increment left and go back to step 1, otherwise go to step 7.

7. If the sum is equal to x, then return true.

PROBLEM 4

What is the smallest positive integer n such that an algorithm whose running time is $100n^2$ runs faster than an algorithm whose running time is 2^n on the same machine. Show your work.

$100n^2$ | 2^n
1: 100 | 2
2: 400 | 4
3: 900 | 8
4: 1600 | 16
5: 2500 | 32
6: 3600 | 64
7: 4900 | 132
8: 6400 | 264
9: 8100 | 512
10: 10000 | 1024
11: 12100 | 2048
12: 14400 | 4096
13: 16900 | 8192
14: 19600 | 16384
15: 22500 | 32768

>> 15

PROBLEM 5

Illustrate on paper the running of the median-finding algorithm (that is, the non-randomized on) discussed in class on the sequence: 8 17 66 2 9 7 3 97 23 79 55 26 8 77 41 1 89 902 46 328 43 20 100 57 62. Drop into your base case (solving the problem “by inspection”) when the length of a sequence is 10 or less.

Find(13, list:
| 8 17 66 2 9 | 7 3 97 23 79 | 55 26 8 77 41 | 1 89 902 46 328 | 43 20 100 57 62 |)

9 23 41 89 57

key = 41

lessList = 8 17 2 9 7 3 23 26 8 1 20
> size = 11
equalList = 41
> size = 1
greaterList = 66 97 79 55 77 89 902 46 328 43 100 57 62
> size = 13

Find(1, greaterList)
| 66 97 79 55 77 | 89 902 46 328 43 | 100 57 62 |

77 89 62

key = 77

lessList = 66 55 46 43 57 62
> size = 6
equalList = 77
> size = 1
greaterList = 97 79 89 902 328 100
> size = 6

Find(1, lessList) = 43

Median = 43

Problems 6-8 use the following definition of *permutation* from the lecture:

List <u>a</u> is a permutation of list <u>b</u> if any of the following are true

- list <u>a</u> and list <u>b</u> are both null, otherwise:
- <u>a.head</u>=<u>b.head</u>, and <u>a.tail</u> is a permutation of <u>b.tail</u>
- <u>a.head</u>≠<u>b.head</u>, and there exists a list <u>c</u> such that
    - <u>a.tail</u> is a permutation of <u>b.head:c</u>, and
    - <u>b.tail</u> is a permutation of <u>a.head:c</u>

<u>PROBLEM 6</u>

Use induction to prove that any (finite) list is a permutation of itself—in other words, that the *permutation* relation is <u>reflexive</u>.

Base case: empty list
Let lst be an empty list. By the definition of *permutation*, lst is a permutation of itself.

Induction Hypothesis: A list of length k-1 is a permutation of itself.
Let <u>lst</u> be a list of length k.
<u>lst</u> will have the same head as itself because it is the same list.
Then, since <u>lst.head</u>=<u>lst.head</u>, in order to prove that <u>lst</u> is a permutation of itself, <u>lst.tail</u> must be a permutation of itself.
The length of <u>lst.tail</u> is one less than the length of <u>lst</u>.
So <u>lst.tail</u> has length k-1.
By the induction hypothesis, <u>lst.tail</u> is a permutation.
Therefore, <u>lst</u> is a permutation by the method of induction.

PROBLEM 7

Using the recursive definition of permutation above, use induction to prove that if list a is a permutation of list b, then list b is a permutation of list a—in other words, that the permutation relation in symmetric.

Base case: a and b are empty lists.
If both a and b are empty lists, then they are the same list.
By the proof from problem 6, a list is a permutation of itself.
Therefore, a and b are permutations of each-other.

Induction Hypothesis: If list a of length k-1 is a permutation of list b of length k-1, then list b is a permutation of list a.
Let lists a and b be length k and let a be a permutation of b.

One of the permutation cases are true…

Case 1: a and b are both empty lists.
As shown by the base case, if a and b are both empty lists, then b is a permutation of a.

Case 2: a.head=b.head, and a.tail is a permutation of b.tail
Because a and b are both length k, then a.tail and b.tail must be of length k-1.
By the induction hypothesis, b.tail is a permutation of a.tail.
Therefore, b is a permutation of a.

Case 3: a.head≠b.head, and there exists a list c such that a.tail is a permutation of b.head:c, and b.tail is a permutation of a.head:c
Since there exists a list c such that b.tail is a permutation of a.head:c and a.tail is a permutation of b.head:c. By the definition of *permutation*, b is a permutation of a.

Therefore, because b is a permutation of a in all possible cases, then b must be a permutation of a if a is a permutation of b.

Then, by the method of induction, if a is a permutation of b, then b is a permutation of a.