# CS 566A2 Midterm Assignment

Due Oct 25 6:00 pm

3

Instructor: Dr. Belyaev

All work must your own. Write your answers on these pages and show your work. If you feel that a question is not fully specified, state any assumptions you need to make in order to solve the problem. Upload your answers on the blackboard to MidTerm Assignment in the CS566_A2 course site under assignments.
**No extensions or late submissions for anything** other than major emergency

Write your name and ID on this page

**Name** Jiankun Dong

**Student ID** U21678916

**Problem 1 [ 20 pts]**

Given this function:

**Function T1:**

```
public int T1( int n ) {
    if (n < 1) {
        return 0;
    } // if
    else {
        return (2*n + T1( n-1 ));
    } // else
} // T1
```

(a) The apperate (2*n) takes ~~constant~~ linear time.

$T_1(0) = T_1(1) = 2$

$T_1(n) = T_1(n-1) + 2n$

$= T_1(n-2) + 2n + 2(n-1)$

$\therefore T_1(n) = \theta(n^2)$

* Assumption: we using effecient algorithm so that 2n calculation is linear time complexity.

**(a)** Set up a recurrence relation for the running time of the function T1 as a function of n. Solve your recurrence relation to specify theta bound of T1.

**Function T2:**

```
public int T2(int n) {
    if (n < 1) {
        return 0;
    } // if
    else {
        return (T1(n) + T2(n-1) – n);
    } // else
} // T2
```

J. D.

b). The opperation (-1) is constant time.

$T_{T_2}(n) = T_{T_1}(n) + T_{T_2}(n-1) + n = T_{T_2}(n-1) + n^2 + n$

$T_{T_2}(1) = 3 + 1 + 1 = 5$

$T_{T_2}(n) = T_{T_2}(n-2) + T_1(n-1) + n^2 + n$

$= (n^2+n) + [(n-1)^2 + (n-1)] + [(n-2)^2 + (n-2)] + \cdots$

$= \frac{n(n+1)(2n+1)}{6} + \frac{n(n+1)}{2} + k$ (some constant that does not involve n.)

$\therefore \Theta(T_2) = n^3$

**(b)** Now set up a recurrence relation for the running time of the function T2 as a function of n. Solve your recurrence relation to specify theta bound of T2.

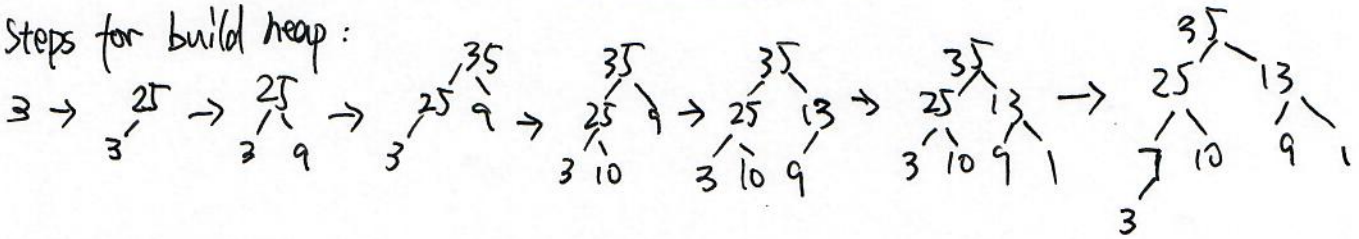
HINT: When doing this, the call to T1 can be replaced by the equation that you found when solving the recurrence relation for T1 in part a).

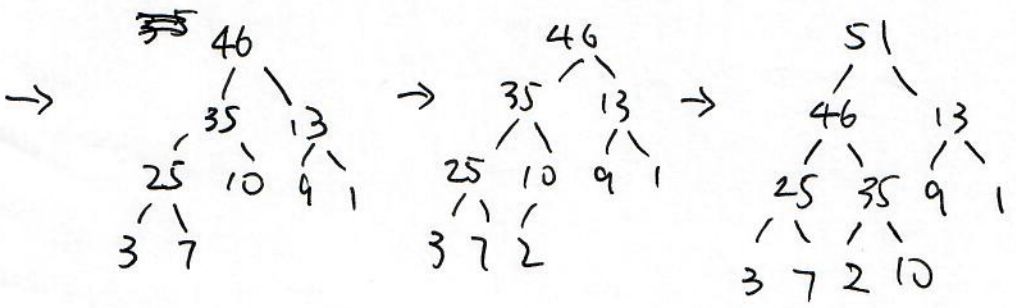
**Problem 2 [15 pts]**

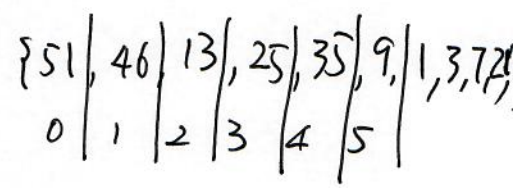
**Assume the complete binary tree numbering scheme used by Heapsort and apply the Heapsort algorithm to the following key sequence (3,25,9, 35,10,13,1,7,46,2,51). The first element index is equal 0.**

**(a) What value is in location 5 of the initial HEAP?**

Steps for build heap:







∴ location 5 ⇒ (9)

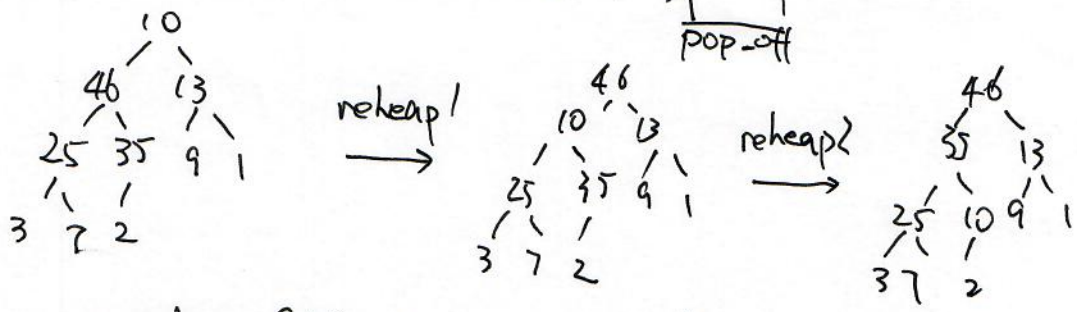
(assuming we start w/ 0)

J.D.

**(b) After a single deletion (of the element at the heap root) and tree restructuring, what value is in location 5 of the new HEAP?**

deletion → {10, 46, 13, 25, 35, 9, 1, 3, 7, 2, | 51 }

pop-off




new heap: {46, 35, 13, 25, 10, | 9 |, 1, 3, 7, 2 }

0 1 2 3 4 | 5 |

It's still (9)

**Problem 3 [20 pts]**

**Assume that we are given n pairs of items as input, where the first item is a number and the second item is one of three colors (red, blue, or yellow). Further assume that the items are sorted by number. Give an O(n) algorithm to sort the items by color (all reds before all blues before all yellows) such that the numbers for identical colors stay sorted.**

**For example: (1,blue), (3,red), (4,blue), (6,yellow), (9,red) should become (3,red), (9,red), (1,blue), (4,blue), (6,yellow).**

```
func sort(A[(number, color)]):
  r = b = y = new list[];
  for i in range(0, len(A)):
      currN = A[i].number
      currC = A[i].color
      switch currC:
          case 'red' : r.append((currN, currC))
          case 'blue' : b.append((currN, currC))
          case 'yellow' : y.append((currN, currC))
  result = new list[]
  result.append(r)
  result.append(b)
  result.append(y)
  Return result.
```

O(n)

O(n)

∴ Overall time complexity O(n)

**Problem 4 [10 pts]**

**You have a computer with only 2Mb of main memory. How do you use it to sort a large file of 500 Mb that is on disk?**

Assumption: we can't use all 2Mb (leave some for sys mem to prevent memory overflow but there's external storage.

Let's seperate the 500Mb to 500 segments. Each Segment contains 1Mb of data.

① First we sort each segment of data w/in it self. Then we write each segment to external storage, repeat for all 500 segments.

② We now read the first 2Kb of data in each sorted external file to a ~~HEAP~~ Heap. for each node in the heap, we store its value, and which segment it came from (path to file). We call this 1Mb of data ~~HeapCurr~~ HeapCurr. This HeapCurr is sorted based on ~~We then write the fis~~ the desired order.

③ ~~Keep~~ We take the ~~node~~ root of the HeapCurr out, read from the root's segment the next value in file, remove it from file and add to HeapCurr (if file empty, we add the Max or Min system value, for minheap and maxheap respectively). Reheapify HeapCurr.

④ Repeat ③ until there's 2Kb of data, ~~write it to~~ append it to the output file. (Or until All node in heap is sys_max/sys_min

⑤ Keep doing ③-④ until all ~~files~~ node in the CurrHeap is sys_max/sys_min. At this point we've sorted all the element in the 500Mb of data.

**Problem 5 [20pts] Insert the keys <13, 19, 35, 71, 31, 6, 23, 4,98> into hash table of size m=13 using linear probing hashing. Here, *h(k, i) =((k mod m) + i) mod m, i=0,1,2,….* How many times you increment *i* to resolve collisions?**

① H(13, 0) = 0

② H(19, 0) = 6

③ H(35, 0) = 9

④ H(71, 0) = 6 ← collision.
H(71, 1) = 7

⑤ H(31, 0) = 5

⑥ H(6, 0) = 6 ← collision
H(6, 1) = 7 ← collision
H(6, 2) = 8

⑦ H(23, 0) = 10

⑧ H(4, 0) = 4

⑨ H(98, 0) = 7 ← collision
H(98, 1) = 8 ← collision
H(98, 2) = 9 ← collision
H(98, 3) = 10 ← collision
H(98, 4) = 11

Total i increase count = 7

Initials: J. D.

**Problem 6 [15 pts] Suppose you have an unsorted array A of n elements and we want to know if A contains any duplicate elements. These elements are integers from the range 1, … , 2n. Tell the asymptotic order T(n) of the worst-case running time for this solution. Try to find the efficient algorithm.**

Assuming: we are more concerned about time than space when defining 'efficient'.

```
func FindDul (A):
    countList = new int [2n+1]          # (index start w/ 0, but we want 1~2n thus 2n+1)
    int i = 0
    for i in range (0, n):              # go through A
        countList [A[i]] += 1           # Count number of time each val appears
        if (countList [A[i]] > 1):
            return True                 # Duplicate detected.
    return False
```

O(n)

---

space: O(n)

time: O(n)

It's better that sorting since that would be O(n log n), but takes more space.

I. D.

**Problem 7 [5pts] ] If you are given a billion integers to sort, what algorithm would you use to sort them? How much time and memory would that consume?**

Assume: we don't know the domain of those int values. If we know the domain and its reasonably sized eg. (int32), then we can use the radix sort.

Thus: Quicksort would be my go to. On large amount of elements, assuming we don't need 'external storage', quicksort on a dataset that is unsorted yields average $O(n \log n) \approx 2.99 \times 10^{10}$ (time)

And $O(\log n)$ space $\approx 29.897$ space complexity.

Something to note is that I'm also assuming that the data is fairly diverse: if we source the data from a relatively small domain, it could greatly affect quicksort's run time.

In that case, we can use radix sort. the exact time and space complexity would depend on the implementation, but in general:

Time $O(d \cdot n)$ d is the digit of largest val expected.

Space $O(n + c)$ c is the amount of time each val can be duplicated.