CS5800: Algorithms — Iraklis Tsekourakis

Homework 2 Name: Avinash R. Arutla

1. (20 points)

1. $T(n) = T(n/2) + T(n/4) + T(n/8) + n$
   Solution:

   Here our guess is T(n) = O(n)

   So to prove that that $T(n) \leq c(n)$

   We can rewrite the equation as,

$$T(n) = T(n/2) + T(n/4) + T(n/8) + n$$
$$T(n) \leq c(n/2) + c(n/4) + c(n/8) + n$$
$$T(n) \leq c(n/2 + n/4 + n/8) + n$$
$$T(n) \leq cn(1/2 + 1/4 + 1/8) + n$$
$$T(n) \leq cn\frac{4+2+1}{8} + n$$
$$T(n) \leq cn\frac{7}{8} + n$$

   To prove tha above statement, we can also write

$$T(n) \leq cn[7/8] + n \leq cn$$
$$cn[7/8] + n \leq cn$$
$$\frac{cn.7 + n(8)}{8} \leq 8cn$$
$$7cn + 8n \leq 8cn$$
$$8n \leq cn$$

   Here we can safely assume that $T(n) \leq 8n$

   We can write this as $c \geq 8$

   So we can adjust our hypothesis by writing c=8

$$T(n) \leq \frac{7}{8}cn + n$$

$$T(n) = \frac{7}{8}8n + n$$

$$T(n) = 8n$$

2. $T(n) = 4T(n/2) + n^2$
Solution:

Here we assume our guess to be $T(n) = O(n^2)$

So we can write the original equation as,

$$T(n) \leq r\frac{cn^2}{2} + n^2$$

$$T(n) \leq cn^2 + n^2$$

$$T(n) \leq (c+1)n^2$$

From the above relation we can understand that it can never be less than cn^2.

We improve our guess to $O(n^2 logn)$.

So we can rewrite this statement as follows

$$T(n) = 4(c\frac{n^2}{4}log\frac{n}{2}) + n^2$$

$$T(n) = 4(cn^2 log\frac{n}{2}) + n^2$$

$$T(n) = n^2(clog(\frac{n}{2}) + 1)$$

We can rewrite this as,

$$n^2(clog(\frac{n}{2}) + 1) \leq cn^2(logn)$$

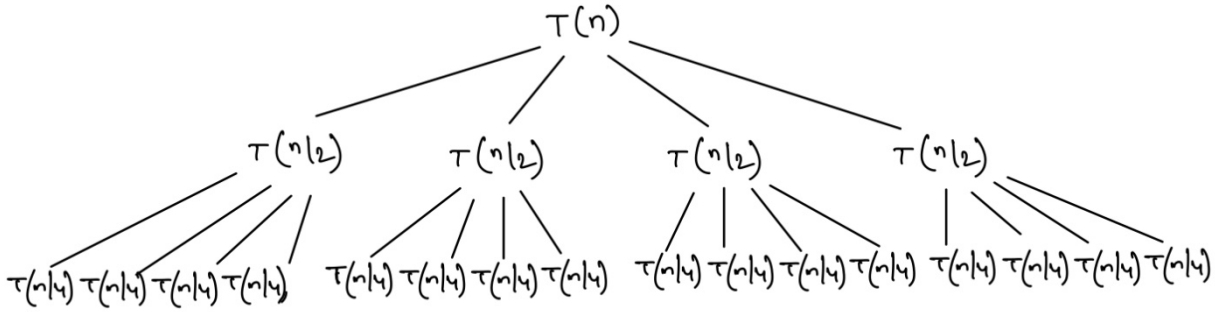
$$clogn - clog2 + 1 \leq cn^2(logn)$$

Here we can say that for a value of $c \geq 0$, and $n \geq n_0$, the $T(n) \leq cn^2 logn$. The $clog^2$ will become obselete as time increases and will therefore satisfy the condition.

2. (20 points)

Solution:

The substitution tree for $T(n) = 4T(n/2) + n$ is as follows



At the end of 4 nodes for each level, the node is being split into 4 parts which will continue for $logn^2$ times.

We assume that our guess is $O(n^2)$.

So we need to prove that $T(n) \leq cn^2$

$$T(n) \leq 4T(n/2) + n$$
$$T(n) \leq 4C(n^2/4) + n$$
$$T(n) \leq \frac{4cn^2}{4} + n$$
$$T(n) \leq n(c+1)$$

Here, we can see that $T(n) \leq cn^2 + n$, here although there is a +n, the cn^2 grows exponentially, so its $O(n^2)$.

3. (20 points)

1. $T(n) = 2T(n/4) + 1$
   Solution:

   Here the equation is in the form $aT(\frac{n}{b} + f(n))$ Where a = 2 and b = 4, and f(n) = 1
   We already know that $f(n) = \theta(n^k log_n^p)$

   So here f(n) = 1, so value of k = 0
   So from master theorem we can say that, since $log_b^a > k$
   The solution would be

   $$\theta(n^(log_b^a))$$

$$\theta(n^(log_4^2))$$

$$\theta(n^{\frac{1}{2}})$$

$$\theta(n^{\sqrt{2}})$$

2. $T(n) = 2T(n/4) + n$
Solution:

Here the equation is in the form $aT(\frac{n}{b} + f(n))$ Where a = 2 and b = 4, and f(n) = n
We already know that $f(n) = \theta(n^k log_n^p)$

So here f(n) = 1, so value of k = 1

So since $k > -1$, according to master theorem the condition would be determined by

$$log_b^a$$

$$log_2^1 = \frac{1}{2}$$

Since $log_b^a < k$, and p =0
So,

$$\theta(n^k log * p_n) = \theta(n)$$

Therefore the tight asymptotic bound here is $\theta(n)$

3. $T(n) = 2T(n/4) + n^2$
Solution:

Here the equation is in the form $aT(\frac{n}{b} + f(n))$ Where a = 2 and b = 4, and f(n) = n^2
We already know that $f(n) = \theta(n^k log_n^p)$

So here f(n) = n^2, so value of k = 2, p =0

$$log_b^a$$

$$log_2^1 = \frac{1}{2}$$

Since $log_b^a < k$, and p =0
So according to master theorem

$$\theta(n^k)$$

$$\theta(n^2)$$

Therefore the tight asymptotic bound here is $\theta(n^2)$

4. $T(n) = 2T(n/4) + n^{\frac{1}{2}}$
Solution:

Here the equation is in the form $aT(\frac{n}{b} + f(n))$ Where a = 2 and b = 4, and f(n) = $\frac{1}{2}$
We already know that $f(n) = \theta(n^k log_n^p)$

So here f(n) = 1, so value of k = $\frac{1}{2}$, p = 0

So, since $log_b^a = k$ and $p = 0$
Hence

$$\theta(n^k log^(p+1)_n)$$

$$\theta(\sqrt{n}logn)$$

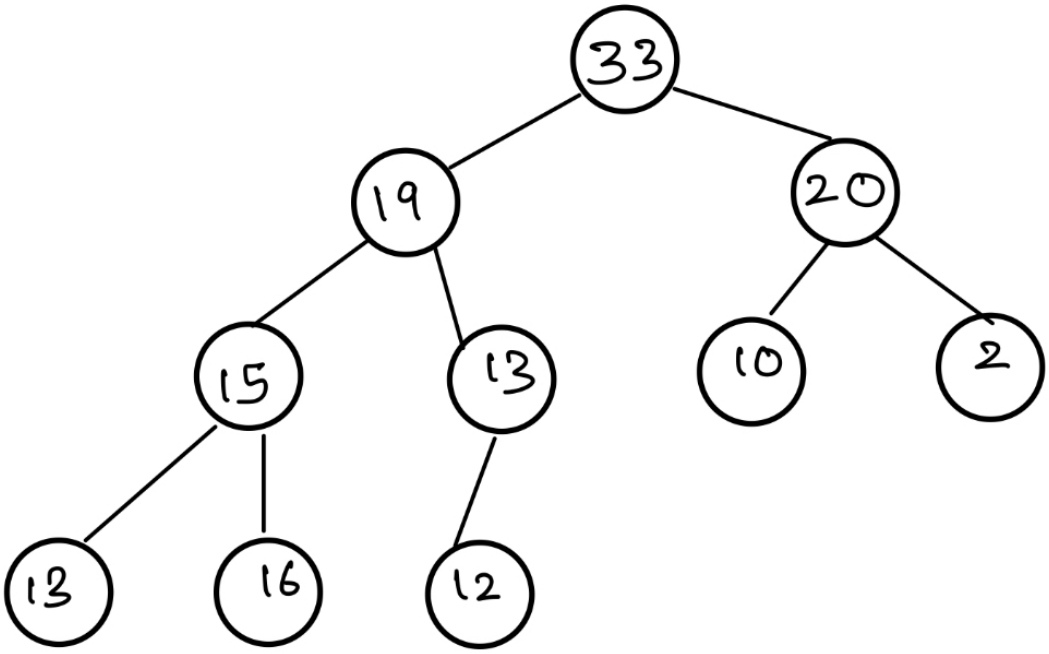
Therefore the tight asymptotic bound here is $\theta(\sqrt{n}logn)$

4. (20 points)

1. Max heap smallest element
Solution:

In a max heap, the smallest element cannot be in a position where it has any children since every parent node must be greater than or equal to every child node, and all elements must be distinct. This is thus because in a max heap, a parent must always be bigger than its offspring by definition. A leaf node must therefore be the smallest element.

2. Is the array with values $(33, 19, 20, 15, 13, 10, 2, 13, 16, 12)$ a max heap?
Solution:



For the max heap property to be satisfied, for a given array, at any element i, the valye of it should be lesser than that of its parents.

As we can observe in this heap, the value of the nodes are mostly similar, but however for node 8, which is in 8th position in the heap array, is greater than its parent 15.

So we can say that this array doesnt satisfy max heap property.

5. (20 points)

Solution:

```
MAX-HEAPIFY(A,i) :
    l = LEFT(i) // returns the index of the left child
    r = RIGHT(i) // returns the index of the right child

    l $\leq$ heap-size(A) and A[l] $>$ A[i]
        largest = l

    r $\leq$ heap-size(A) and A[r] $>$ A[i]
        largest = r
```

```
if largest!=i
    swap A[i] with A[largest]
    l = largest
else
    break
```

6. (15 points)

Solution:

We aim to merge k sorted lists using a min heap. The process is as follows:

1. Initialize the heap with the first element of each array.
2. While the heap is not empty, pop the minimum element from the heap and add it to the final array.
3. If the popped element has a successor in its original list, insert the successor into the heap.
4. Repeat this process until the heap is empty.

The pseudo-code for the above logic is given below:

```
len_lists = len(lists)
heap = MinHeap()
final_array = []

# Initialize the heap with the first element of each array
for i in range(0, len_lists):
    if lists[i]:  # Checking if the list is not empty
        heap.insert((lists[i][0], i, 0))

# Extracting the elements in sorted order
while not heap.isEmpty():
    value, index, element_index = heap.popMin()
    final_array.append(value)
    if element_index + 1 < len(lists[index]):
        next_element = lists[index][element_index + 1]
        heap.insert((next_element, index, element_index + 1))
```

This method ensures that we efficiently merge k sorted lists by always choosing the smallest current element available among the lists.