COMP2123-Assignment 5

May 24, 2024

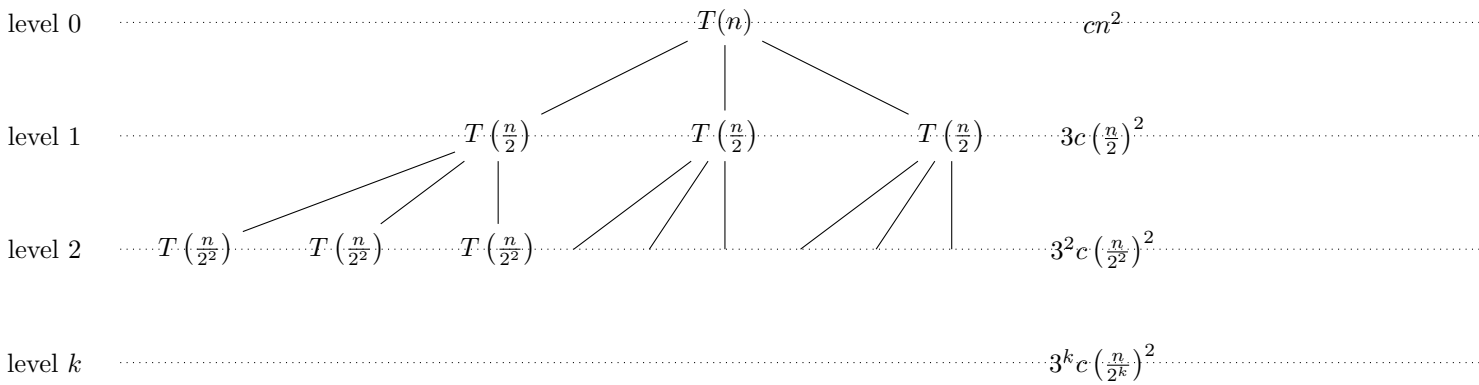
Notation Clarification

This section is to clarify the notations used throughout this assignment.

$[a : b]$	The sequence a, $a + 1, \ldots, b - 1$.
`int`	The data type representing integers.
`real`	The data type representing the real numbers.
`void`	Used to show that a function does not return anything.
`null`	The variable representing nothingness.
`bool`	The data type representing a boolean value which is either true or false.
`!`	Logical expressions and performing logical negation in programming

Problem 1: Recurrence Relations Analysis

a) $T(n) = 3T(n/2) + n^2$



- Since we halve the size every time, there are totally $k = \log_2 n$ levels. Summing all this up, we get:

$$T(n) = cn^2 + 3c\left(\frac{n}{2}\right)^2 + 3^2c\left(\frac{n}{2^2}\right)^2 + 3^3c\left(\frac{n}{2^3}\right)^2 + \ldots + 3^kc\left(\frac{n}{2^k}\right)^2 \tag{1}$$

$$= cn^2\left(3^0\left(\frac{1}{2^0}\right)^2 + 3^1\left(\frac{1}{2}\right)^2 + 3^2\left(\frac{1}{2^2}\right)^2 + 3^3\left(\frac{1}{2^3}\right)^2 + ... + 3^k\left(\frac{1}{2^k}\right)^2\right) \tag{2}$$

$$= cn^2\left(3^0\left(\frac{1}{2^2}\right)^0 + 3\left(\frac{1}{2^2}\right) + 3^2\left(\frac{1}{2^2}\right)^2 + 3^3\left(\frac{1}{2^2}\right)^3 + ... + 3^k\left(\frac{1}{2^2}\right)^k\right) \tag{3}$$

$$= cn^2\left(1 + \left(\frac{3}{4}\right) + \left(\frac{3}{4}\right)^2 + \left(\frac{3}{4}\right)^3 + \ldots + \left(\frac{3}{4}\right)^k\right) \tag{4}$$

$$= cn^2\left(\sum_{i=0}^{\log_2 n}\left(\frac{3}{4}\right)^i\right) \tag{5}$$

- When we apply the bound on the geometric series to (5) with $r = \frac{3}{4}$, we get:

$$T(n) = cn^2\left(\frac{1 - \left(\frac{3}{4}\right)^{\log_2 n+1}}{1 - \frac{3}{4}}\right) \tag{6}$$

$$= cn^2\left(4\left(1 - \left(\frac{3}{4}\right)^{\log_2 n+1}\right)\right) \tag{7}$$

$$= cn^2\left(4 - 4 \cdot \frac{3}{4} \cdot \left(\frac{3}{4}\right)^{\log_2 n}\right) \tag{8}$$

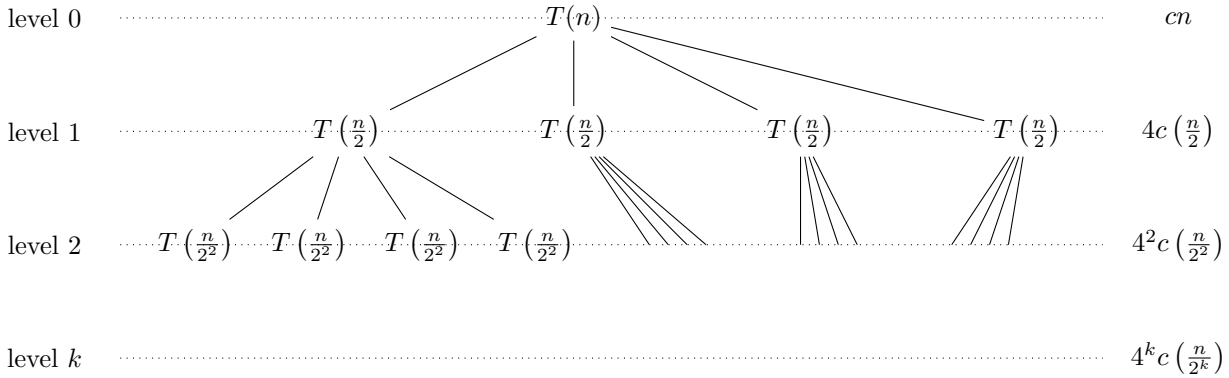
$$= cn^2\left(4 - 3 \cdot \left(\frac{3}{4}\right)^{\log_2 n}\right) \tag{9}$$

- Using the formula $a^{\log_2 b} = b^{\log_2 a}$, we have:

$$\begin{aligned} T(n) &= cn^2 \left(4 - 3 \cdot (n)^{\log_2 \frac{3}{4}}\right) \\ &= cn^2 \left(4 - 3 \cdot (n)^{\log_2 3 - 2}\right) \\ &= cn^2 \left(4 - 3 \cdot \left(\frac{n^{\log_2 3}}{n^2}\right)\right) \\ &= 4cn^2 - 3cn^2 \cdot \frac{n^{\log_2 3}}{n^2} \\ &= 4cn^2 - 3cn^{\log_2 3} \end{aligned}$$

Since $\log_2 3$ is a constant, the term $3cn^{\log_2 3}$ is much smaller than $4cn^2$ as n grows larger. Therefore, the dominant term is $4cn^2$, and we can conclude that $T(n) = \Theta(n^2)$.

b) $T(n) = 4T(n/2) + n$



- Since we halve the size every time, there are totally $k = \log_2 n$ levels. Summing all this up, we get:

$$T(n) = cn + 4c\left(\frac{n}{2}\right) + 4^2 c\left(\frac{n}{2^2}\right) + 4^3 c\left(\frac{n}{2^3}\right) + \ldots + 4^k c\left(\frac{n}{2^k}\right) \tag{10}$$

$$= cn\left(\left(\frac{4}{2}\right)^0 + \left(\frac{4}{2}\right)^1 + \left(\frac{4}{2}\right)^2 + \left(\frac{4}{2}\right)^3 + \ldots + \left(\frac{4}{2}\right)^k\right) \tag{11}$$

$$= cn(1 + 2 + 2^2 + 2^3 + \ldots + 2^k) \tag{12}$$

- When we apply the bound on the geometric series to (12) with $r = 2$, we get:

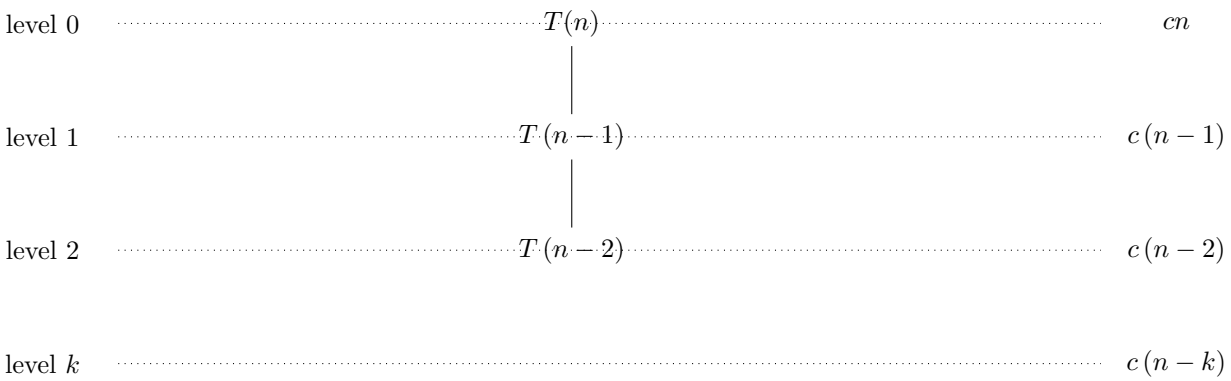
$$T(n) = cn(\frac{1-2^{k+1}}{1-2}) \tag{13}$$
$$= cn(2^{k+1} - 1) \tag{14}$$
$$= cn\left(2 \cdot 2^k - 1\right) \tag{15}$$
$$= 2cn \cdot 2^{log_2 n} - cn \tag{16}$$
$$\tag{17}$$

- Using the formula $a^{\log_2 b} = b^{\log_2 a}$, we have:

$$T(n) = 2cn \cdot n^{log_2 2} - cn \tag{18}$$
$$= 2cn^2 - cn \tag{19}$$

the term cn is much smaller than $2cn^2$ as n grows larger. Therefore, the dominant term is $2cn^2$, and we can conclude that $T(n) = \Theta(n^2)$.

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



- Since we reduce the size by 1 every time, there are totally $k = n - 1$ levels. Summing all this up, we get:

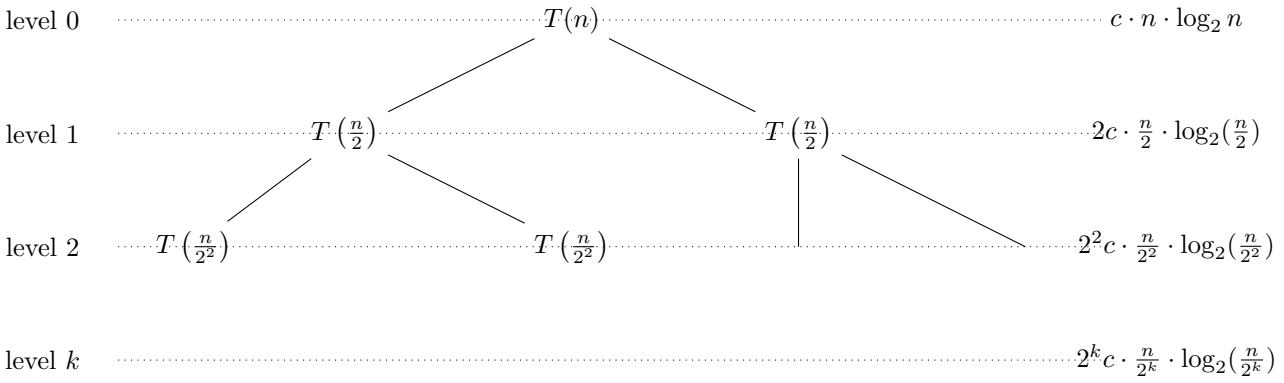
$$T(n) = cn + c(n-1) + c(n-2) + c(n-3) + ... + c(n-k) \tag{20}$$
$$= c(n + (n-1) + ... + 3 + 2 + 1) \tag{21}$$

- We have formula $1 + 2 + 3 + ... + n = \frac{n(n+1)}{2} for$ (21), we have:

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

The term $\frac{1}{2}cn$ is much smaller than $\frac{1}{2}cn^2$ as n grows larger. Therefore, the dominant term is $\frac{1}{2}cn^2$, and we can conclude that $T(n) = \Theta(n^2)$.

d) $T(n) = 2T(n/2) + n \log n$



- Since we halve the size every time, there are totally $k = \log_2 n$ levels. Summing all this up, we get:

$$T(n) = cn \cdot \log_2 n + 2c \cdot \frac{n}{2} \cdot \log_2(\frac{n}{2}) + 2^2 c \cdot \frac{n}{2^2} \cdot \log_2(\frac{n}{2^2}) + ... + 2^k c \cdot \frac{n}{2^k} \cdot \log_2(\frac{n}{2^k}) \tag{22}$$

$$= cn \left(log_2 n + 2\frac{1}{2} \log_2(\frac{n}{2}) + 2^2 \frac{1}{2^2} \log_2(\frac{n}{2^2}) + ... + 2^k \frac{1}{2^k} \log_2(\frac{1}{2^k}) \right) \tag{23}$$

$$= cn \left(log_2 n + log_2(\frac{n}{2}) + log_2(\frac{n}{2^2}) + ... + log_2(\frac{n}{2^k}) \right) \tag{24}$$

$$= cn \left((log_2 n - log_2 2^0) + (log_2 n - log_2 2) + (log_2 n - log_2 2^2) ... + (log_2 n - log_2 2^k) \right) \tag{25}$$

$$= cn \left((k+1) log_2 n - (log_2 2^0 + log_2 2^1 + log_2 2^2 + ... + log_2 2^k) \right) \tag{26}$$

$$= cn \left(k log_2 n + log_2 n - (0 log_2 2 + 1 log_2 2 + 2 log_2^2 + ... + k log_2 2) \right) \triangleright \textit{ Using Power rule of logarithms} \tag{27}$$

$$= cn \left(k log_2 n + log_2 n - (0 + 1 + 2 + ... k) \right) \tag{28}$$

$$= cn \left(k log_2 n + log_2 n - \frac{k(k+1)}{2} \right) \tag{29}$$

$$= cn \left((log_2 n)^2 + log_2 n - \frac{log_2 n (log_2 n + 1)}{2} \right) \tag{30}$$

$$= cn \left((log_2 n)^2 + log_2 n - \frac{(log_2 n)^2 + log_2 n}{2} \right) \tag{31}$$

$$= cn (log_2 n)^2 + cn log_2 n - \frac{1}{2} cn (log_2 n)^2 - \frac{1}{2} cn (log_2 n) \tag{32}$$

$$= \frac{1}{2} cn (log 2_n)^2 - \frac{1}{2} cn (log_2 n) \tag{33}$$

$$\tag{34}$$

The term $\frac{1}{2} cn (log_2 n)$ is much smaller than $\frac{1}{2} cn (log 2_n)^2$ as n grows larger. Therefore, the dominant term is $\frac{1}{2} cn^2$, and we can conclude that $T(n) = \Theta(n (log_2 n)^2)$.

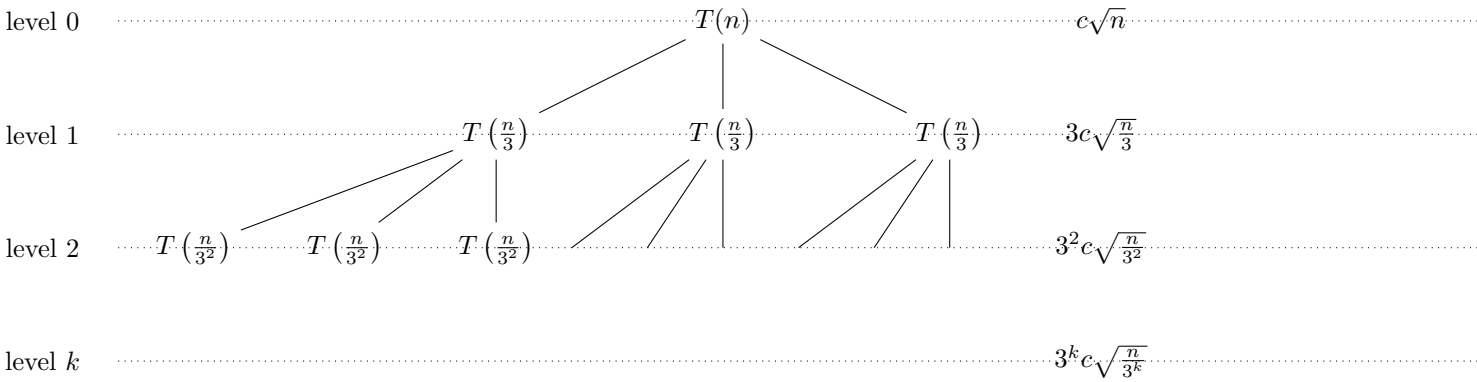
e) $T(n) = \sqrt{2}T(n/2) + \log n$

In this case, we have:

$$n^{\log_b a} \text{ is } n^{\log_2 \sqrt{2}} \text{ which is } n^{1/2}$$

And we also have, $f(n) = \log_2 n$. Since $\log_2 n$ is asymptotically less than any positive number of n $f(n)$ is $(n^{\log_b a - \epsilon})$ for any $0 < \epsilon < \frac{1}{2}$. Therefore, by Master Theorem, $T(n)$ is $\Theta(\sqrt{n})$ in this case

f) $T(n) = 3T(n/3) + \sqrt{n}$



- Since we reduce the size by a factor of three each time, there are totally $k = \log_3 n$ levels. Summing all this up, we get:'

$$T(n) = c\sqrt{n} + 3c\sqrt{\frac{n}{3}} + 3^2c\sqrt{\frac{n}{3^2}} + ... + 3^kc\sqrt{\frac{n}{3^k}} \tag{35}$$

$$= c\sqrt{n}\left(3^0\sqrt{\frac{1}{3^0}} + 3^1\sqrt{\frac{1}{3^1}} + 3^2\sqrt{\frac{1}{3^2}} + ... + 3^k\sqrt{\frac{1}{3^k}}\right) \tag{36}$$

$$= c\sqrt{n}\left(\sqrt{\frac{(3^0)^2}{3^0}} + \sqrt{\frac{(3^1)^2}{3^1}} + \sqrt{\frac{(3^2)^2}{3^2}} + ... + \sqrt{\frac{(3^k)^2}{3^k}}\right) \tag{37}$$

$$= c\sqrt{n}\left(\sqrt{3^0} + \sqrt{3^1} + \sqrt{3^2} + ... + \sqrt{3^k}\right) \tag{38}$$

$$= c\sqrt{n}\left((3^{(1/2)})^0 + (3^{(1/2)})^1 + (3^{(1/2)})^2 + ... + (3^{(1/2)})^k\right) \tag{39}$$

- When we apply the bound on the geometric series to (39) with $r = 3^{1/2}$, we get:

$$
\begin{aligned}
T(n) &= c\sqrt{n}\frac{(3^{1/2})^{k+1}-1}{3^{1/2}-1} && (40)\\
&= c\sqrt{n}\frac{(3^{1/2})^{\log_3 n+1}-1}{3^{1/2}-1} \triangleright\ k=\log_3(n) && (41)\\
&= c\sqrt{n}\frac{\sqrt{3}(3^{1/2})^{\log_3 n}-1}{3^{1/2}-1} && (42)\\
&= c\sqrt{n}\frac{\sqrt{3}(n)^{\log_3 3^{1/2}}-1}{3^{1/2}-1} && (43)\\
&= c\sqrt{n}\frac{\sqrt{3}(n)^{1/2}-1}{\sqrt{3}-1} && (44)\\
&= c\sqrt{n}\frac{\sqrt{3n}-1}{\sqrt{3}-1} && (45)\\
&= \frac{\sqrt{3}c}{\sqrt{3}-1}\cdot\sqrt{n}\cdot\sqrt{n}-\frac{c\sqrt{n}}{\sqrt{3}-1} && (46)\\
&= \frac{\sqrt{3}c}{\sqrt{3}-1}\cdot n-\frac{c\sqrt{n}}{\sqrt{3}-1} && (47)\\
& && (48)
\end{aligned}
$$

The term $\frac{c\sqrt{n}}{\sqrt{3}-1}$ is much smaller than $\frac{\sqrt{3}cn}{\sqrt{3}-1}$ as n grows larger. Therefore, the dominant term is $\frac{\sqrt{3}cn}{\sqrt{3}-1}$, and we can conclude that $T(n)=\Theta(n)$.

g) $T(n)=7T(n/3)+n^2$

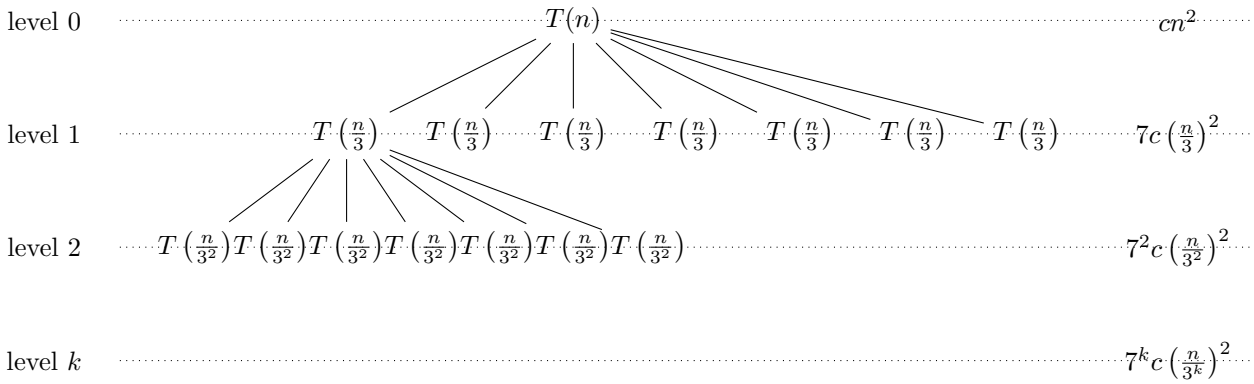


Figure 1: Recurrence tree for $T(n)=7T(n/3)+n^2$

- Since we reduce the size by a factor of three each time, there are totally $k=\log_3 n$ levels. Summing all this

up, we get:

$$T(n) = cn^2 + 7c\left(\frac{n}{3}\right)^2 + 7^2c\left(\frac{n}{3^2}\right)^2 + 7^3c\left(\frac{n}{3^3}\right)^2 + \ldots + 7^kc\left(\frac{n}{3^k}\right)^2 \tag{49}$$

$$= cn^2\left(7^0\left(\frac{1}{3^0}\right)^2 + 7^1\left(\frac{1}{3}\right)^2 + 7^2\left(\frac{1}{3^2}\right)^2 + 7^3\left(\frac{1}{3^3}\right)^2 + ... + 7^k\left(\frac{1}{3^k}\right)^2\right) \tag{50}$$

$$= cn^2\left(7^0\left(\frac{1}{3^2}\right)^0 + 7\left(\frac{1}{3^2}\right) + 7^2\left(\frac{1}{3^2}\right)^2 + 7^3\left(\frac{1}{3^2}\right)^3 + ... + 7^k\left(\frac{1}{3^2}\right)^k\right) \tag{51}$$

$$= cn^2\left(1 + \left(\frac{7}{9}\right) + \left(\frac{7}{9}\right)^2 + \left(\frac{7}{9}\right)^3 + \ldots + \left(\frac{7}{9}\right)^k\right) \tag{52}$$

$$= cn^2\left(\sum_{i=0}^{\log_3 n}\left(\frac{7}{9}\right)^i\right) \tag{53}$$

- When we apply the bound on the geometric series to (53) with $r = \frac{7}{9}$, we get:

$$T(n) = cn^2\left(\frac{1 - \left(\frac{7}{9}\right)^{\log_3 n+1}}{1 - \frac{7}{9}}\right) \tag{54}$$

$$= cn^2\left(\frac{9}{2}\left(1 - \left(\frac{7}{9}\right)^{\log_3 n+1}\right)\right) \tag{55}$$

$$= cn^2\left(\frac{9}{2} - \frac{9}{2}\cdot\frac{7}{9}\cdot\left(\frac{7}{9}\right)^{\log_3 n}\right) \tag{56}$$

$$= cn^2\left(\frac{9}{2} - \frac{7}{2}\cdot\left(\frac{7}{9}\right)^{\log_3 n}\right) \tag{57}$$

- Using the formula $a^{\log_3 b} = b^{\log_3 a}$, we have:

$$T(n) = cn^2\left(\frac{9}{2} - \frac{7}{2}\cdot(n)^{\log_3 \frac{7}{9}}\right)$$

$$= cn^2\left(\frac{9}{2} - \frac{7}{2}\cdot(n)^{\log_3 7-2}\right)$$

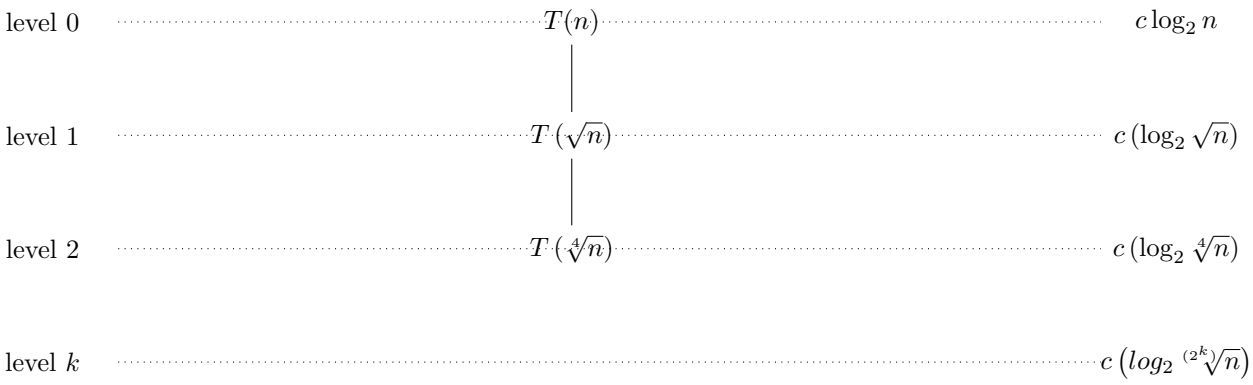
$$= cn^2\left(\frac{9}{2} - \frac{7}{2}\cdot\left(\frac{n^{\log_3 7}}{n^2}\right)\right)$$

$$= \frac{9}{2}cn^2 - \frac{7}{2}cn^2\cdot\frac{n^{\log_3 7}}{n^2}$$

$$= \frac{9}{2}cn^2 - \frac{7}{2}cn^{\log_3 7}$$

Since $\log_3 7$ is a constant, the term $\frac{7}{2}cn^{\log_3 7}$ is much smaller than $\frac{9}{2}cn^2$, and we can conclude that $T(n) = \Theta(n^2)$.

h) $T(n) = T(\sqrt{n}) + \log n$



- From $T(n) = T(\sqrt{n}) + \log_2 n$ we can also write as:

$$T(n) = T(n^{1/2}) + \log_2 n$$

- This does not show a constant where we can stop. Therefore we need to subtitute n (Let $n = 2^m$), we have:

$$T(2^m) = T(2^{m/2}) + \log_2(2^m) \tag{58}$$
$$= T(2^{m/2}) + m \triangleright \textit{ Using Power rule} \tag{59}$$
$$\tag{60}$$

- Let $S(m) = T(2^m)$ so $S(m/2) = T(2^{m/2})$. Thus, we have:

$$S(m) = S(m/2) + m \tag{61}$$

- Now we will find the complexity of (61)
  - Since we halve the size everytime, there are totally $k = \log_2 m$ levels. Summing all this up,we get:

$$S(m) = c(\frac{m}{2^0}) + c(\frac{m}{2}) + c(\frac{m}{2^2}) + ... + c(\frac{m}{2^k}) \tag{62}$$
$$= cm\left((\frac{1}{2})^0 + (\frac{1}{2})^1 + (\frac{1}{2})^2 + ... + (\frac{1}{2})^k\right) \tag{63}$$

– When we apply the bound on the geometric series to (63) with $r = \frac{1f}{2}$, we get:

$$S(m) = cm\left(\frac{(\frac{1}{2})^{k+1} - 1}{\frac{1}{2} - 1}\right) \tag{64}$$

$$= -2cm\left(\left(\frac{1}{2}\right)^{\log_2(m)+1} - 1\right) \tag{65}$$

$$= -2cm\left(\frac{1}{2}\left(\frac{1}{2}\right)^{\log_2 m} - 1\right) \tag{66}$$

$$= -2cm\left(\frac{1}{2}m^{\log_2(\frac{1}{2})} - 1\right) \tag{67}$$

$$= -2cm\left(\frac{1}{2m} - 1\right) \tag{68}$$

$$= -c + 2cm \tag{69}$$

$$\tag{70}$$

– $2cm$ is a dominant term. Thus, the complexity of $S(m) = \Theta(m)$

- We started with the recurrence $T(n) = T(\sqrt{n}) + \log_2 n$ and transformed n to 2^m, which led us to a new recurrence $S(m) = S(m/2) + m$, where $S(m) = T(2^m)$. At the same time, complexity of $S(m) = \Theta(m)$ and $m = \log_2 n$, substituting back, we get:

$$S(\log_2 n) = \Theta(\log_2 n)$$

- By our transformation definition $S(m) = T(2^m)$. Setting $m = \log_2 n$ implies $2^m = n$. Thus, we have:

$$S(\log_2 n) = T(n)$$

- Therefore, substituting our expression for $\log_2 n$:

$$T(n) = \Theta(\log_2 n)$$

Problem 2

2.1 Combine two top Skeleton as an input and produce single top Skeleton in $O(n)$

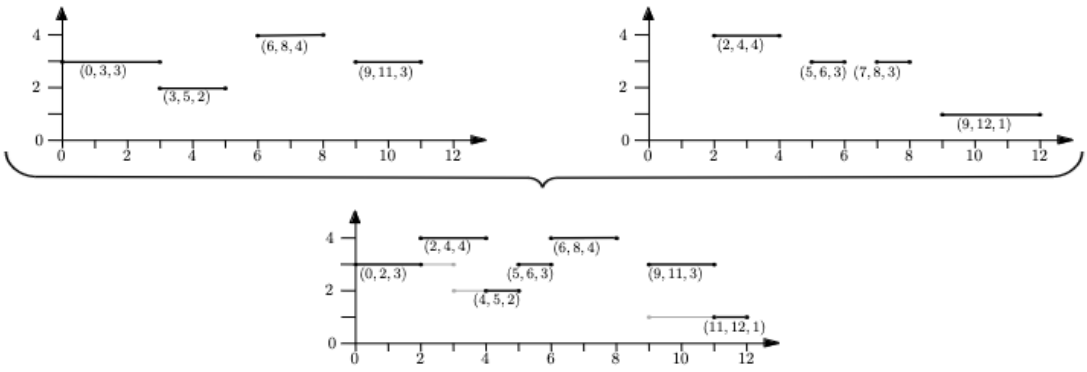
(a) **Description of algorithm:**

(a) Firstly, we have two dynamic arrays, which are the top skeleton of the first top skeleton and second top skeletonsH_1 and H_2.

- I call H_1 is the dynamic array store top skeleton in the first top skeleton, which being sorted by horizontal segments ordered from left to right
- I call H_2 is the dynamic array store top skeleton in the second top skeletons, which being sorted by horizontal segments ordered from left to right

(b) Thus, when I access each segment in both top skeletons, it will cost $\Theta(1)$ time, and each segment in both H_1 and H_2 is the array of size three with the first and second elements are left and right x-coordinates of the segment respectively and the third element denotes the y-coordinate of that segment.

(c) With the example of the image belows we will have the following input



- In this example, we will have inputs:

$$H_1 = \{\{0, 3, 3\}, \{3, 5, 2\}, \{6, 8, 4\}, \{9, 11, 3\}\} \tag{71}$$
$$H_2 = \{\{2, 4, 4\}, \{5, 6, 3\}, \{7, 8, 3\}, \{9, 12, 3\}\} \tag{72}$$

- And the expected output will be an dynamic array calls **result**. Thus, each time we add new element in that dynamic array will have the complexity is $\mathcal{O}(1)$ amortized time.
- For this example, the output will be

$$result = \{\{0, 2, 3\}, \{2, 4, 4\}, \{4, 5, 2\}, \{5, 6, 3\}, \{6, 8, 4\}, \{9, 11, 3\}, \{11, 12, 1\}\} \tag{73}$$

(d) The pseudocode of function `CombineSkeletons(H1, H2)`, after having the pseudocode I will explain how it works. Before writting the pseudocode I will have some function, which I will use in the pseudocode

- `min(a, b)`: return the minimum value of integer a and integer b (The complexity is $O(1)$)
- `max(a, b)`: return the maximum value of integer a and integer b (The complexity is $O(1)$)

- `append(segment)`: append a segment into dynamic array. Because I use dynamic array to store and append always import new element at the end of the list so the complexity will be $O(1)$

```
function CombineSkeletons(H_1, H_2)
    H = []                                                          ▷ Final TopSkeletons
    i, j = 0, 0                                                     ▷ Shift indices for H1 and H2
    while i < length of H_1 and j < length of H_2 do
        Array segment_1 ← H_1[i]
        Array segment_2 ← H_2[j]                                    ▷ Comparing two segments from H_1 and H_2 to solve collapse
        if segment_1[1] ≥ segment_2[0] and segment_1[1] ≥ segment_2[0] then   ▷ case when collapsed
            min_start ← min(segment_1[0], segment_2[0])
            max_start ← max(segment_1[0], segment_2[0])
            min_end ← min(segment_1[1], segment_2[1])
            max_end ← max(segment_1[1], segment_2[1]) ▷ Find unique point two split two segments into three
            intervals, which are first, second and third part
            first_part ← []
            third_part ← []
            if segment_1[0] < segment_2[0] then                     ▷ Case for finding the height of first interval
                first_part ← [min_start, max_start, segment_1[2]]
            else
                first_part ← [min_start, max_start, segment_2[2]]
            second_part ← [max_start, min_end, max(segment_1[2], segment_2[2])]   ▷ Second interval
            if segment_1[1] > segment_2[1] then                     ▷ Case for finding the height of third interval
                third_part ← [min_end, max_end, segment_1[2]]
            else
                third_part ← [min_end, max_end, segment_2[2]]
     ▷ After having three different interval, now I will check that the case two segments will have the that interval
    or not, if not starting x-cor and ending x-cor will be the same
            if first_part[0] != first_part[1] then                  ▷ If we exist the first_part, add it to final dynamic array
                H.append(first_part)
            if second_part[0] != second_part[1] then ▷ If we exist the second_part, add it to final dynamic array
                H.append(second_part)
                                    ▷ Now we will check the last part, which will need to add back to H_1 or H_2
            if segment_1[1] >segment_2[1] then ▷ segment_1 extends beyond the end of segment_2, which means
            the third part will be back in H_1
                if third_part[0] != third_part[1] then              ▷ check the existance of third part
                    H_1[i] = third_part
                j ← j + 1
            else if segment_1[1] <segment_2[1] then                 ▷ the third part will be in H_2
                if third_part[0] != third_part[1] then              ▷ check the existance of third part
                    H_1[i] = third_part
                i ← i + 1
            else          ▷ The case that they have same endpoint, thus we will skip to next index in both array
                j ← j + 1
                i ← i + 1
        else                                                        ▷ If they are not collapse, find correct segment and put it into H
            if segment_1[0] < segment_2[0] or (segment_1[0] == segment_2[0] and segment_1[1] ≤ segment_2[1])
            then                                                    ▷ This case, segment1 will be appended to H
                H.append(segment_1)
                i ← i + 1
            else
                H.append(segment_2)
                j ← j + 1
    while i < length(H_1) do                                        ▷ Add remaining segments
        H.append(H_1[i])
        i ← i + 1
    while j < length(H_2) do
        H.append(H_2[j])
        j ← j + 1
    return H
```


Algorithm 1: Combine Skeletons

(e) As in the example, the output of the `CombineSkeletons` is

$$H = \{\{0, 2, 3\}, \{2, 3, 4\}, \{3, 4, 4\}, \{4, 5, 2\}, \{5, 6, 3\}, \{6, 7, 4\}, \{7, 8, 4\}, \{9, 11, 3\}, \{11, 12, 1\}\}$$

(f) As we can see this output is correct with H is a top Skeleton. However, some segments can be merged with other segments. Thus, to make the output shorter like the output of template, I will have another methods, which calls `merging(H)` run in $O(n)$ time

```
1: function MERGING(H)
2:     result ← []
3:     for each segment seg in H do
4:         if merged_H is not empty and last element in merged_H ends at the same point where seg starts and
           last element in merged_H has the same height as seg then
5:             Update the end point of the last segment in merged_H to the end point of seg
6:         else
7:             Add seg to merged_H
8:     Return merged_H
```

Algorithm 2: Merging algorithm

(g) using `merging` function I will have expected output, which is

$$H = \{\{0, 2, 3\}, \{2, 4, 4\}, \{4, 5, 2\}, \{5, 6, 3\}, \{6, 8, 4\}, \{9, 11, 3\}, \{11, 12, 1\}\}$$

(h) **Description how my algorithms works**

- Firstly, I will initialise a new dynamic array to store the result of combining two topSkeletons (final merged segments) and I Initialize indices i and j to 0. These will iterate H_1 and H_2 respectively.
- After that, I will have the main loop, and there loop will continue looping as long as there are segments left in both H_1 and H_2. In the loop, I Retrieve the current segments from H_1 and H_2 using indices i and j. Then, I will check the segments are overlap or not. This is done by checking if the end of one segment is greater than or equal to the start of the other and vice versa.
  - **Case 1: Two segments does not overlap** ▷ *It will go to first else confition in Pseudocode of CombineSkeleton*
    - **Segment from H_2 extends beyond H_1**: Append *segment_1* to H and increase the indices i for H_1 by 1.
    - **Segment from H_1 extends beyond H_2**: Append *segment_2* to H and increase the indices i for H_2 by 1.
  - **Case 2: Two segments overlaps** ▷ *It will go to the first if condition in Pseudocode*
    - When segments s1 and s2 overlap, the algorithm will divide the segment into thre parts, which calls first_part, second_part and third_part. Thus, to splits into three parts, I have to find significant points using `min` and `max` functions ▷ *From line 8 to 11 in Pseudocode*
      - `min_start`: the left-most point of the overlap
      - `max_start`: the second left-most point of the ovdrlap
      - `min_end`: the third left-most point of the overlap
      - `max_end`: the right-most point of the overlap

    * From all significant points of the overlap, I will have three part. ▷ *from line 12 to line 22 in Pseudocode*
      · `first_part`: will be the array of size 3, which is {`min_start`, `max_start`, **y-coordinate of the segment that have min_start**}
      · `second_part`: will be the array of size 3, which is {`max_start`, `min_end`, **y-coordianate of the segment that have max y-coordinate**}
      · `third_part`: will be the array of size 3, which is {`min_end`, `max_end`, **y-coordianate of the segment that have max_end**}
    * After we have three part, I will check `first_part`: ▷ *From line 23 to 24 in Pseudocode*
      · If the first part, if start point is the sam as end point, I will not add that part to H, because when they have the same start and end point, which means two segments at this case when overlap does not have first part. Otherwise, I will add `first_part` to H
    * for the `second_part`, I will do the same as I did for `first_part` ▷ *from line 25 to 26 in Pseudocode*
    * For the `third_part`, I will do differently with the first two part ▷ *From line 27 to 37*
      · First, I will use if condition to check which segment extends beyond the other segment. For example segment_1 extends beyond segment_2, I will change value of H_1 at index i to the third part and increase j by 1. Other case like segment_2 extends beyond segment_1, I will change value of H_2 at index j to the third part and increas i by 1. At the case two of them are equals, which means we do not have third part. Thus, I increase both i and j by 1
- Finally, I will have to while-loop for H_1 and H_2 to add remaining segments
- To make the result like expected output, I will call `merging`.
  – This function will create a new dynamic array calls result.
  – After that, it will start for-loop to terate over each segment "*seg*". In that loop, I will have condition for merging, which are;
    * Check if the merged_H list is not empty.
    * Check if the end point of the last segment in `merged_H` is equal to the start point of the current segment
    * Check if the height of the last segment in merge_H s the same as the height of the current segment
  – **Merging segment**: If all the conditions are met, update the end point of the last segment in merged_H to the end point of the current segment
  – **Adding New Segment**: If any of the conditions are not met, Add the current segment to merged_H

(b) **Argue the correctness**

- The algorithm correctness can be showed by utilizing loop invariant.
  – **Invariant**: At the beginning of every iteration of the while loop, the list H comprises the accurately processed segments considered thus far from both H_1 and H_2, with H_1 and H_2 being two top skeletons containing sorted segments.
  – **Initialize**: At the start, before any iterations occur, the algorithm has not yet examined any segments from H_1 and H_2. As a result, H_1 and H_2 are unchanged, and H are empty. Therefore, the invariant is valid at this point. (It is important to note that both H_1 and H_2 are TopSkeletons)
  – **Maintenance**

* During each iteration, the algorithm will take the two segment at the value of H_1 at index i and H_2 at index j. After have that segments, the algorithm will have two cases, which is two segments are overlap or not by using conditional statement ▷ *The condition is in line 7*
* **Case 1: Overlap**
  · After finding all the significant points, I will have three parts. With the first_part that have starting coordinate at `min_start` and end at `end_start`. Thus, that part will node overlap with other segments. The reason for that is because it will end at the start of other segments.
  · For the second_part, it will starts at max_start and end at min_end. Thus, this part can be an overlap parts if this part exists. That is why the algorithm will remove the overlap part by choosing the part with higher y-coordinate. Not only that, because we choose only part with higher y-coordinate and skip append the part with lower coordinate, so after append this part to H, we do not have any overlap part in H. Thus, the invariant still holds
  · In term of the third_part, because third_part is the sub part of segment_1 or segment_2 depending on different cases, when we add back to H_1 or H_2 depending on the part is the sub part of segment_1 and segment_2. Thus, we do not have any overlap segments in H_1 and H_2 after add the third_part back to index i or j. The reason for that is because before adding third_part to H_1 or H_2, H_1 and H_2 are TopSkeletons, which means when I change that segment to sub part of that segment, H_1 or H_2 also be the TopSkeletons. Thus, the invariant still holds
* **Case 2: Not overlap**
  · The algorithm picks the segment having ealier starting position element from two non-overlapping segments to include in set H H. This ensures that only correct segments are added to set H. Thus preserving the invariance.
* After that, I will add every remaining segments of H_1 or H_2 into H. Because we have solved every overlap possible, all remaining segments is part of TopSkeletons. Thus, the loop invariant still holds
* Taking all those into consideration, we can conclude that after each iteration, the invariant holds

– **Termination**: The loop will terminate when we scan every segments in both H_1 and H_2 at least once. Thus, we will have H is the combine Top skeletons of H_1 and H_2. Thus, the invariant stills holds at termination

- In conclusion, by maintaining the loop invariant in all iterations, the correctness of the algorithm is guaranteed.

(c) **Prove an upper bound on the time complexity of your algorithm**:

- **Initialisation**: At the start, I initialise H equals to empty dynamic array. After that, I initialise i and j equals to 0. Thus, the complexity of this one is $\mathcal{O}(1)$
- **First While loop**: Let a is the size of H_1 and b is the size of H_2. The main while loop will have $a + b$ iterations. Each iteration will run in $\mathcal{O}(1)$ time, which being proved belows
  – In each iteration, the algorithm will retrieve the current segments from `H_1` and `H_2` using indices i and j. Thus, the complexity of this one is $\mathcal{O}(1)$ ▷ *in line 5 and 6*
  – After that, I will have the first condtional statement to check two segments are overlap or not in $O(1)$ time. ▷ *in line 7 of Pseudocode*

* If the condtion are true it will run in line 8 to line 37 in Pseudocode. From line 8 to line 13, all of this lines will run in O(1) time becasue it just merely assigning using `min` and `max` function, which is being proved run in O(1) time before. In line 14, 19, 23, 25, 27, 31 in Pseudocode, we will have seperate conditional statements, which run in $\mathcal{O}(1)$ time. This is mainly because it just a logical operations. In each conditional statements, It run in $\mathcal{O}(\infty)$ time because inside each conditional statements, It just increase i or j by 1 or using `append` function or changing the value in H_1 or H_2 by using index. All of these will run in $O(1)$ time. Thus inside in each iteration, it will run in constant time, which is $\mathcal{O}(1)$
* If the condition are not true, it will run in line 39 to 44. Because the condtion statement in line 39 run in $O(1)$ and inside that conditional statement it also run in $O(1)$ time (`append` runs in O(1) time). Thus, in each iteration, it will run in $O(1)$ time.

– After that, I will have two while loop which runs in $O(a)$ and $O(b)$. This is because in side them, the algorithm use only `append` function and increase indicies by one. Thus both of them run in $O(a)$ or $O(b)$ time

- The complexity of this algorithm is:

$$O(1) + O(1) + O(a+b) + O(a) + O(b) = \mathcal{O}(a+b) = \mathcal{O}(n) \quad \blacksquare$$

2.2: Divide-and-Conquer algorithm that computes the top skeleton of a set S of n horizontal line segments in the plane

a **Description how my algorithms works**

```
1: function TopSkeletons(S)
2:     if Size of s larger than 1 then
3:         mid ← int(size of s / 2)
4:         top1 ← TopSkeletons(ls[:mid])
5:         top2 ← TopSkeletons(ls[mid:])
6:         return CombineSkeleton(top1, top2)
7:     else
8:         return S
```

Algorithm 3: Wrong algorithm

- My algorithm uses Divide and Conquer, so I will explain it in two steps: dividing and conquering.
- **Dividing and Conquere**
  – My algorithm uses recursion to divide the original skeletons until the size of the skeletons equals 1, which means the size of an input to the `TopSkeletons` function equals 1. First, I need a conditional statement to check if the size of the input is larger than 1 ▷ *Line 2 of the pseudocode.* If the size equals 1, I will stop the dividing step and return that array. If the size of the input is larger than 1, I will assign `mid` to half the number of segments in S, and use recursion to assign `top1` and `top2`.
- **Combining**
  – After dividing the original skeletons into subproblems, the algorithm proceeds to combine the results. Once `top1` and `top2` are computed by recursively calling `TopSkeletons` on the divided segments,

the `CombineSkeleton` function is called to merge these results ▷ *Line 5 of the pseudocode.* This function ensures that the combined skeleton maintains the properties required by the algorithm.

- The `CombineSkeleton` function takes `top1` and `top2` as inputs and merges them, ensuring that no incorrect segments are included in the combined result. The specifics of this merging process depend on the criteria defined by the problem, such as maintaining order or ensuring no overlaps.
- Finally, the combined skeleton is returned as the result of the `TopSkeletons` function call. By ensuring that each recursive step correctly processes and combines the segments, the algorithm maintains its correctness and efficiency.

Overall, this Divide and Conquer approach effectively breaks down the problem into smaller, manageable parts, solves each part recursively, and then combines the results to form the final solution.

b **Argue the correctness**

- We prove this algorithm's correctness via induction
- Base case: when the input has 1 or 0 segment, that input is trivially the top skeleton of itself sorted from left to right, returning the correct result.
- Recursive case: If we have calculated the top skeletons of the two halves correctly, we can find the top skeletons sorted from left to right of both halves combined correctly. This is ensured by the correctness of our CombineSkeletons() algorithm.
- Addendum: the program always reaches its base case as by dividing the input list in half repeatedly, we will always reach the base case where the original list has been divided into lists with less than 2 segments within.
- As the base case is correct and the recursive case maintains this correctness, we can conclude that this algorithm is correct.

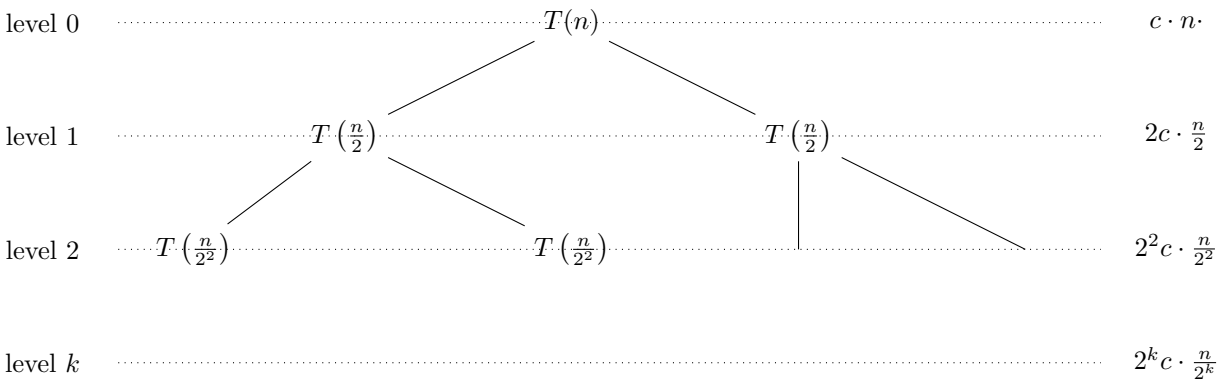
c **Analysis the complexity**

- In the base case, the algorithm simply returns the list itself, which takes $O(1)$ time. The divide step also runs in $O(1)$ time because it only involves calculating the midpoint and slicing the original list. The recursive step computes the top skeleton for each half recursively, and the combine step takes $O(n)$ time for n segments. Thus, we have the following recurrence relation:

$$T(n) = \begin{cases} T(n) = 2T(n/2) + O(n) & \text{for } n > 1 \\ O(1) & \text{for } n \leq 1 \end{cases}$$

d Thus, I will have the following chart then I will prove using unrolling

e Since we halve the size every time. Thus, the level is $k = \log_2 n$. Summing all this up, we get



$$T(n) = cn + 2c\frac{n}{2} + 2^2 c\frac{n}{2^2} + ... + 2^k c\frac{n}{2^k} \tag{74}$$

$$= cn\left((\frac{2}{2})^0 + (\frac{2}{2})^1 + (\frac{2}{2})^2 + ... + (\frac{2}{2})^k\right) \tag{75}$$

$$= cn\left(1^0 + 1^1 + 1^2 + ... + 1^k\right) \tag{76}$$

$$\tag{77}$$

Because k $= \log_2 n$, $\left(1^0 + 1^1 + 1^2 + ... + 1^k\right) = \log_2 n$

Thus, we have:

$$T(n) = 2\log_2 n$$

Thus, the complexity of the algorithm is $\mathcal{O}(n \log_2 n)$