CS204: Design and Analysis of Algorithms

220001007, 220001009

January 12, 2024

1 Insertion Sort

```
for i=2 to A.length
    key = A[i]
    j=j-1
    while j>0 & A[i]>key
        A[i+1] = A[i]
A[i+1] = key
```

The cost of <while condition> would be $\sum_{i=2}^{A.length} t_j$**,** where t_j for $j = 2, 3, ..., n$ denotes the number of times the while loop test is executed for that value of i.

2 Theta (θ) Notation

$$\theta(g(n)) = f(n) \mid \exists\, n_0 \in \mathbb{Z}^+,\ c_1, c_2 \in \mathbb{R}^+, \text{ such that}$$
$$0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)\ \forall\, n \geq n_0$$

and

$$f(n) = \theta(g(n)) \text{ iff } f(n) = O(g(n)) \text{ and } f(n) = \Omega(g(n))$$

2.1 Example

To show

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

Solution:

We can write

$$\frac{n^2}{2} - (\frac{n}{2}.\frac{n}{2}) \leq \frac{n^2}{2} - \frac{n}{2} \leq \frac{n^2}{2}$$
$$\frac{n^2}{4} \leq \frac{n^2}{2} - \frac{n}{2} \leq \frac{n^2}{2}$$

and so for n_0

$$\frac{n_0^2}{4} \leq \frac{n_0^2}{2} - \frac{n_0}{2} \quad \text{and} \quad \frac{n_0^2}{2} - \frac{n_0}{2} \leq \frac{n_0^2}{2}$$
$$\frac{n_0}{2} \leq \frac{n_0^2}{4} \quad \text{and} \quad \frac{n_0}{2} \geq 0$$
$$n_0 \geq 2 \quad \text{and} \quad n_0 \geq 0$$

Therefore,

$$c_1 = \frac{1}{4} \text{ and } c_2 = \frac{1}{2} \text{ for } n_0 \geq 2$$

3 Strict Bounds

3.1 o-Notation

Definition:

$$o(g(n)) = \{f(n) \mid \exists\ n_0 \in \mathbb{Z}^+,\ \text{and}\ \forall\ c \in \mathbb{R}^+ \text{such that } 0 \leq f(n) < c.g(n)\ \forall\ n \geq n_0\}$$

and

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$$

3.2 ω-Notation

Definition:

$$\omega(g(n)) = \{f(n) \mid \exists\ n_0 \in \mathbb{Z}^+,\ \text{and}\ \forall\ c \in \mathbb{R}^+ \text{such that } 0 \leq c.g(n) < f(n)\ \forall\ n \geq n_0\}$$

and

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$$

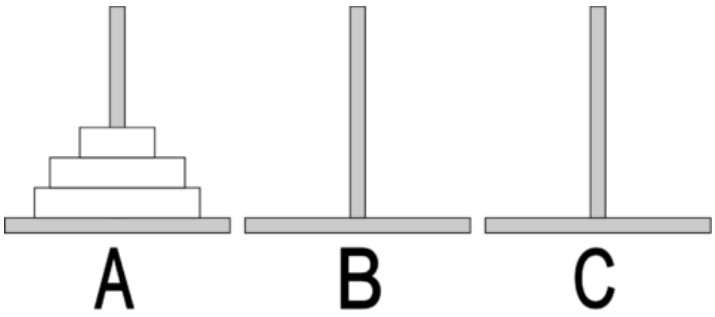
4 Space Complexity

4.1 Insertion Sort and Merge Sort

- Insertion Sort is an in-place sorting algorithm, meaning it does not require additional memory space proportional to the input size. The space complexity of Insertion Sort is O(1), indicating constant space usage.
- Merge Sort works by recursively dividing the input array into smaller halves, sorting them, and then merging them back together. Merge Sort typically requires additional space for the merging process, and this additional space is proportional to the input size. The space complexity of Merge Sort is O(n) because it requires additional space to store the merged sub-arrays.

5 Towers of Hanoi Problem

Move all the disks from A to B, where only 1 disk can be moved at a time and no larger disk can sit on smaller disk.



Solution:

- move n disks from A to B
- move n-1 disks from A to C
- move 1 disk from A to B
- move n-1 disk from C to B

Now, let

$$M(n) = \text{number of moves required to transfer n disks from one peg to another}$$

$$\therefore M(n) = M(n-1) + 1 + M(n-1)$$

$$M(1) = 1 \qquad \text{(Base Case)}$$

Using the method of Repetitive Substitution,

$$\begin{aligned}
M(n) &= 2M(n-1) + 1 \\
&= 2[2M(n-2) + 1] + 1 \\
&= 2^2(M(n-2)) + 2 + 1 \\
&= 2^3(M(n-3)) + 4 + 2 + 1 \\
&\cdot \\
&\cdot \\
&= 2^j M(n-j) + 2^{j-1} + 2^{j-2} + \ldots\ldots + 2 + 1 && (jsteps) \\
&\cdot \\
&\cdot \\
&= 2^{n-1}M(1) + 2^{n-2} + 2^{n-3} + \ldots\ldots + 2 + 1 && (nsteps) \\
&= 2^{n-1}.1 + 2^{n-2} + 2^{n-3} + \ldots\ldots + 2 + 1 && (M(1) = 1)
\end{aligned}$$

As this is GP,

$$M(n) = 2^n - 1 \quad = O(2^n)$$

Therefore, the time complexity of the problem is $O(2^n)$.

6 Master's theorem

Definition: For a ≥ 1, b ≥ 1 and a non-negative function $f(n)$

$$T(n) = aT(n/b) + f(n) \text{ and } T(1) = \theta(1)$$

then

$$T(n) = \theta(n^{\log_b a}) + \sum_{j=0}^{\log_b n - 1} a^j f(\frac{h}{b^j})$$

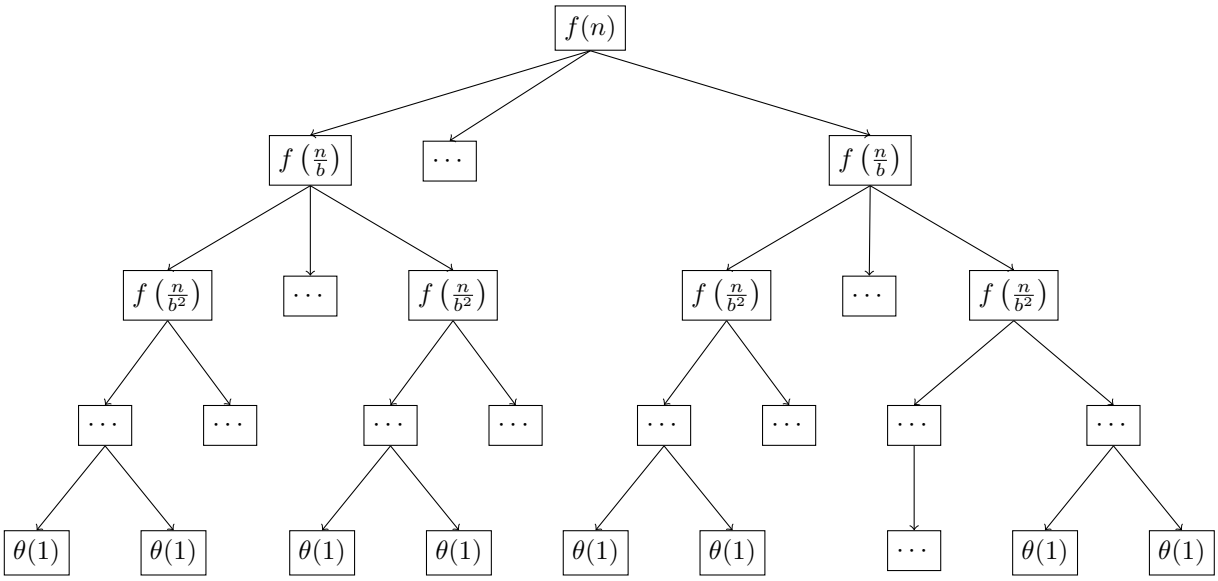
and

- If there exists a constant $\epsilon > 0$ such that $f(n) = O(n^{\log_b a - \epsilon})$, then $T(n) = \theta(n^{\log_b a})$.
- If $f(n) = \theta(n^{\log_b a})$, then $T(n) = \theta(n^{\log_b a} log\ n)$.
- If there exists a constant $\epsilon > 0$ such that $f(n) = \Omega(n^{\log_b a + \epsilon})$ and $af(n/b) \leq cf(n)$ for $c < 1$, then $T(n) = \theta(f(n))$.

Proof: We have

$$\begin{aligned}
T(n) &= aT(n/b) + f(n) \\
&= a\,[\,a\,T(\frac{n}{b^2}) + f(\frac{n}{b})\,] + f(n)
\end{aligned}$$

The Recursion Tree would be



- Depth of the tree: $\lfloor log_b n \rfloor$
- No. of Leaves: $a^{\lfloor log_b n \rfloor}$
- Leaves $\{\theta(1)\}$: $a^{\log_b n}\ \theta(1) = \theta(n^{\log_b a})$ (because, $n^{\log_b a} = a^{\log_b n}$)
- Cost at particular level of the tree:
  - Level 0: $a^0 f(n/b^0)$
  - Level 1: $af(n/b)$
  - Level 2: $a^2 f(n/b^2)$
  - .
  - Level j: $a^j f(n/b^j)$
  - .
  - At last level: $a^{\log_b n}\ f(n/b^{\log_b n}\ )$

Total Cost $T(n) =\ af(n/b)+a^2f(n/b^2)+...+a^jf(n/b^j)+....+a^{\log_b n-1}\ f(\frac{n}{b^{\log_b n-1}})+a^{log_b n}f(n/b^{log_b n})$

$$\therefore\ \ T(n) = \sum_{j=0}^{\log_b n-1} a^j f(\frac{n}{b^j}) + a^{\log_b n} f(\frac{n}{n}) \qquad (\text{as} \quad b^{\log_b n} = n)$$

$$= \sum_{j=0}^{\log_b n-1} a^j f(\frac{n}{b^j}) + a^{\log_b n} f(1)$$

$$= \sum_{j=0}^{\log_b n-1} a^j f(\frac{n}{b^j}) +\ \theta(n^{\log_b a})$$

$$\text{Let}\ \ g(n) = \sum_{j=0}^{\log_b n-1} a^j f(\frac{n}{b^j})$$

Case 1: When $f(n) = O(n^{\log_b a - \epsilon})$ for some $\epsilon > 0$

$$\begin{aligned}
g(n) &= \sum_{j=0}^{\log_b n - 1} a^j f(\frac{n}{b^j})^{log_b a - \epsilon} \\
&= n^{log_b a - \epsilon} \sum_{j=0}^{\log_b n - 1} (\frac{ab^\epsilon}{b^{log_b a}})^j \\
&= n^{log_b a - \epsilon} \sum_{j=0}^{\log_b n - 1} (\frac{ab^\epsilon}{a})^j \\
&= n^{log_b a - \epsilon} \sum_{j=0}^{\log_b n - 1} (b^\epsilon)^j \\
&= n^{log_b a - \epsilon} \ \frac{b^{\epsilon \log_b n - 1}}{b^\epsilon - 1} \\
&= n^{\log_b a - \epsilon} (\frac{n^\epsilon - 1}{b^\epsilon - 1}) \qquad (\text{as } b^{\log_b n} = n) \\
g(n) &= O(n^{\log_b a})
\end{aligned}$$

Therefore, $T(n)$ would be $\theta(n^{\log_b a}) + O(n^{\log_b a})$

$$\therefore \quad T(n) = \theta(n^{\log_b a})$$

Case 2: When $f(n) = \theta(n^{\log_b a})$

$$\begin{aligned}
g(n) &= \sum_{j=0}^{\log_b n - 1} a^j f(\frac{h}{b^j})^{log_b a} \\
&= n^{log_b a} \sum_{j=0}^{\log_b n - 1} (\frac{a}{b^{log_b a}})^j \\
&= n^{log_b a} \sum_{j=0}^{\log_b n - 1} (\frac{a}{a})^j \\
&= n^{log_b a} \sum_{j=0}^{\log_b n - 1} (1)^j \\
g(n) &= \theta(n^{\log_b a} log\, n)
\end{aligned}$$

Therefore, $T(n)$ **would be** $\theta(n^{\log_b a} log\, n)$**.**

Case 3: When $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some $\epsilon > 0$ and $af(n/b) \leq cf(n)$ for $c < 1$.

Since we have $af(n/b) \leq cf(n)$, using induction we can write that for j iterations would yield

$$a^j(f(\frac{n}{b^j})) \leq c^j(f(n))$$

Now

$$
\begin{aligned}
g(n) &= \sum_{j=0}^{\log_b n} a^j f(n/b^j) \\
&\leq \sum_{j=0}^{\log_b n} c^i f(n) \\
&= f(n) \sum_{j=0}^{\log_b n} c^i \\
&\leq f(n) \sum_{j=0}^{\infty} c^i \qquad \text{.....(for sufficiently large n)} \\
&= f(n) \frac{1}{c-1} \\
\therefore \quad g(n) &= \theta(f(n))
\end{aligned}
$$

Therefore, $T(n)$ **would be** $\theta(f(n))$.