Nafis Abeer

EC 330 HW 3

1) a) Master thm. $T(n) = a\,T(n/b) + \Theta(n^d)$

$$T(n) = \begin{cases} \Theta(n^d) & a < b^d \\ \Theta(n^d \log n) & a = b^d \\ \Theta(n^{\log_b a}) & a > b^d \end{cases}$$

A: $a = 4$ $b = 2$ $d =$ linear so $\Theta(n)$

$T(n) = 4\,T(n/2) + \Theta(n)$ $\log_b a = \log_2 4 = 2$

$4 > 2^1$ so $a > b^d$ so $T(n) = \Theta(n^{\log_b a}) \Rightarrow$ $T(n) = \underline{\Theta(n^2)}$

B: $A = 2$ size $n-1$ $d = 0$ (constant

$T(n) = 2T(n-1) + \Theta(k)$ $T(n-1) = 2T(n-2) + \Theta(k)$

$\hookrightarrow T(n) = 2(2T(n-2) + \Theta(k)) + \Theta(k)$ $T(n-2) = 2T(n-3) + \Theta(k)$

$= 2(2(2T(n-3) + \Theta(k)) + \Theta(k)) + \Theta(k)$

$\cdots\ 2^3 T + 2^2(k+k) + k$

$= 2^3 T(n-3) + 7k$ $k = n \rightarrow$ goes until $n - k = 0$

$\hookrightarrow 2^k T(n-k) + k(2^k - 1)$

$T(n) = 2^n T(0) + k(2^n - 1)$

$= 2^n[T(0) + k] - k \rightarrow T(n) = \underline{\Theta(2^n)}$

C: $a = 9$ size: $n/3 \Rightarrow b = 3$ $d = 2$ as time complexity: $O(n^2)$

$T(n) = 9\,T(n/3) + O(n^2)$

$9 = 3^2$ so $a = b^d \Rightarrow T(n) = (n^d \log n)$ $T(n) = \underline{O(n^2 \log n)}$

b) • $T(n) = 5\,T(n/3) + n^3$ master theorem applicable : $A = 5$ $b = 3$ $d = 3$

$A = 5 < 3^3 \Rightarrow a < b^d \Rightarrow T(n) = \Theta(n^d) = \underline{\Theta(n^3)}$

• $T(n) = 2T(n/4) + 3n^{1/2}$ master theorem applicable: $A = 2$ $b = 4$ $d = 1/2$

$A = b^d \Rightarrow 2 = 4^{1/2} \rightarrow T(n) = O(n^d \log n) = O(\underline{\sqrt{n}\log n})$

$T(n) = \underline{O(\log n!)}$

• $T(n) = T(n-1) + \log n$ master theorem does not apply $T(1) + \log n! = T(n)$

$T(n-1) = T(n-2) + \log(n-1)$

$T(n) = T(n-2) + \log(n-1) + \log(n)$ $T(n) = O(\log n!)$

$T(n) = T(n-2) + \log n(n-1) = T(n-3) + \log n \cdots (n-2)$

$T(n-2) = T(n-3) + \log(n-2)$ $\cdots$ $T(1) + \log n(n-1)(n-2)\ldots 1 = T(1) + \log n! = T(n)$

- $T(n) = n\left(T\left(\frac{n}{2}\right)\right)^3$

$T\left(\frac{n}{2}\right) = \frac{n}{2}\left(T\left(\frac{n}{2\cdot 2}\right)\right)^3$

$T\left(\frac{n}{2^k}\right) = \frac{n}{2^k}\left(T\left(\frac{n}{2^{k+1}}\right)\right)^3$

$\frac{n}{2^{k+1}} = 1 \quad n = 2^{k+1} \quad \log_2 n = k$

$\log_2 n - 1 = k$

$\rightarrow T(n) = n\left(\frac{n}{2}\left(T\left(\frac{n}{2\cdot 2}\right)\right)^3\right)^3$

$\rightarrow k=1$

$T(n) = \frac{1}{2^3} \cdot n \cdot n^3 \cdot T\left(\frac{n}{2^2}\right)^{3^3}$

$\downarrow k=1$

$T(n) = \text{constant} \cdot n \cdot n^{3^k} \cdot T\left(\frac{n}{2^{k+1}}\right)^{3\cdot 3^k}$

when this is $T\left(\frac{n}{2^{k+1}}\right) = T(1)$

$\therefore \quad T(n) = n^{3^{\log_2 n - 1}+1} \cdot (1)$ some power

$\cancel{T(n) = (n^{3\log_2 n+1})}$

$T(n) = n^{3(\log_2 n - 1)+1}$

$T(n) = O\left(n^{3\log_2 n - 2}\right)$

$T(n) \cong O(n^n)$

- $T(n) = T\left(\frac{n}{2}\right) + 2^n$

$T\left(\frac{n}{2}\right) = T\left(\frac{n}{2^2}\right) + 2^{n/2}$

$T\left(\frac{n}{2^k}\right) = T\left(\frac{n}{2^{k+1}}\right) + 2^{n/2^k}$

$T\left(\frac{n}{2^{k+1}}\right) \Rightarrow T(1) = 1$

$\frac{n}{2^{k+1}} = 1 \quad n = 2^{k+1} \quad k = \log_2 n - 1$

$T\left(\frac{n}{2^2}\right) = T\left(\frac{n}{2^{2+1}}\right) + 2^{n/2^2}$

$T(n) = T\left(\frac{n}{2^{k+1}}\right) + \underbrace{2^{n/2^2} + 2^{n/2^1} + 2^n}_{\text{will happen } \log n \text{ times}}$

$\frac{n}{2^{k+1}} = 1$

$T(n) = T(1) + 2^n\left(2^{\frac{1}{k+1}} + 2^{\frac{1}{k}} + 2^{\frac{1}{k-1}} \ldots \log n \text{ times}\right)$

$= 2^n \log n$

$T(n) = O\left(2^n \log n\right)$

$\rightarrow T(n) = T\left(\frac{n}{2^2}\right) + 2^{n/2} + 2^n$

$\downarrow k=1$

$T(n) = T\left(\frac{n}{2^{k+1}}\right) + 2^{n/2^k} + 2^n$

when this is 1

$T(1) + 2^{n/2^k} + 2^{n/2^{k-1}} \ldots + 2^{n/2^0}$

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

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

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

$\cancel{T(n) = O(2^n)}$

a)
2)
```
int i = n;
while (i>1) {
      int j = i;
      while (j<n) {
            int k = 0;
            while (k<n) {
                  k = k+10;
            }
            j = j * 2;
      }
      i = i / 2;
}
```

`while (k<n) { k = k+10; }` → runs $n/10$ times → $O(n)$

`j = j * 2;` → runs until $2^m = n$ (m times) $= \log_2 n = m$ → $\log n$ times

`i = i / 2;` → runs until 2^m gives us n since $\frac{i}{2^m}$ has to equal 1 and $i = n$ → runs $\log n$ times

Total time complexity: $O(n) \cdot O(\log n) \cdot O(\log n) = O(n \log^2 n)$

b)
```
void StrangeSort(int a[], int min, int max) {
      if (min >= max)
            return;
      if (a[min] > a[max])
            swap (a[min], a[max]); // constant time      + O(1)
      int one_third = (max-min+1) / 3;        -> divide into 3
      if (one-third >= 1) {
            strangesort(a[], min, max - onethird);    -> size 2/3 of original
            strange sort(a[], min + one-third, max);  -> size 2/3 of original
            strange sort(a[], min, max - one-third);  -> size 2/3 of original
      }
}
```

$T(n) = 3\left(\frac{2}{3}n\right) + O(1)$

master theorem:

$a = 3 \quad b = \frac{3}{2} \quad d = 0$

$\log_b a = \log_{\frac{3}{2}} 3 = 2.7095$

$a > \frac{3}{2}^0$ so $\theta(n) = O(n^{\log_b a}) = O(n^{2.7095})$

$O(n^{2.7095})$