ARYAMAN MISHRA — 19BCE1027

1

```
procedure bubbleSort (list : array of items)
  loop = list. count;  — 1
  for i=0 to loop-1 do:
    swapped = false
  for j=0 to loop-1 do
    /* compare adjacent elements */
    if list[j] > list[j+1] then
    /* swap them */
      temp = list[j];  — 1
      list[j] = list[j+1];  — 1
      list[j+1] = temp;  — 1
      swapped = true  — 1
    end if
  end for
 /* if no number swapped, array is sorted,
      break the loop */
   if (not swapped) then
    break  — 1
   end if
  end for
 end procedure
 return lst
 main ()
 {
  initialize array  — 1
  input elements in array
  ~~bubblesort (array);~~
  initialize x as int array
     x[] = bubblesort (array);  — 1
   a = x[array.length - 1];  — 1
   b = x[array.length - 2];  — 1
   print a and b;  — 2
 }
```

Step count

} $(\text{length})^2$

$(n-1)$ comparisons in 1st pass,
$(n-2)$ " 2nd pass
$(n-3)$ in 3rd pass
with n being number of elements to be sorted

} length (for-loop)

Step count = $n^2 + (n-1)! + 6 + n + 5$

$n \rightarrow$ number of elements

Worst Case Time Complexity : $O(n^2)$ (Big-O)

Best Case Time Complexity = $O(n^2)$ (Big-omega) (Ω)

Average Case Time Complexity = $O(n^2)$ (Big Theta) (Θ)

Space Complexity : $O(1)$

$O = \Theta(n^2)$

$(n-1)(n-2) + (n-3) + \dots + 3 + 2 + 1$

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

2. a) $f(n) = n - 100$

$g(n) = n - 200$

$O(n)$, $\therefore f = \Theta(g(n))$

b) $f(n) = 100n + \log n$

$g(n) = n + (\log n)^2$

$$\lim_{n\to\infty} \frac{100n + \log n}{n + (\log n)^2}$$

$$\lim_{n\to +\infty} \frac{100 + \frac{1}{n}}{1 + \frac{2\log n}{n}} \approx 100 \Rightarrow f(n) = \Theta(g(n))$$

c) $f(n) = \log 2n$

$g(n) = \log 3n$

$f(n) = \log 2 + \log n$

$g(n) = \log 3 + \log n$

$f(n) = \Theta(g(n))$

d) $f(n) = n^{1.01}$

$g(n) = n \log^2 n$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \cancel{\frac{n^{1.01}}{n\log^2 n}} = \cancel{\lim_{n\to\infty}} \cancel{\frac{1.01\, n^{-99}}{\log^2 n}}$$

$$\lim_{n\to\infty} \frac{n^{1.01}}{n\log^2 n} = \lim_{n\to\infty} \frac{n^{0.01}}{\log^2 n} = \lim_{n\to\infty} \frac{0.01\, n^{-0.99}}{\frac{2\log n}{n}}$$

$$\lim_{n\to\infty} \frac{0.005\, n^{0.01}}{\log n} = \lim_{n\to\infty} \frac{0.005 \times 0.01 \times n^{0.01-1}}{\frac{1}{n}}$$

$$\Rightarrow \lim_{n\to\infty} 0.05 \times 0.01 \times n^{0.01} \Rightarrow f(n) = \Omega(g(n))$$

e) $f(n) = \cancel{2n}\ n2^n$
$g(n) = 3^n$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \cancel{n2^n}\ \frac{n2^n}{3^n} = \frac{n}{(1.5)^n}$$

$$\lim_{n\to\infty} \cancel{\frac{n.2^n \log 2 + 2^n.1}{3^n \log 3}} \quad \cancel{\lim_{n\to\infty} \frac{n}{(1.5)^n}}$$

$$\lim_{n\to\infty} \cancel{n.\left(\frac{2}{3}\right)^n} \Rightarrow \cancel{\lim_{n\to\infty} 1.\left(\frac{2}{3}\right)^n \log\left(\frac{2}{3}\right) + \left(\frac{2}{3}\right)^n}$$

$n = O((1.5)^n)$

$f = O(g(n))$

f) $f(n) = n!$
$g(n) = 2^n$

~~$f(n) = \theta$~~

value of $n! \Rightarrow n! > \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$

$n! = \Theta(n \log n)$

$f = \Omega(g)$ states that when comparing 2 functions $f(n)$ and $g(n)$, $f(n)$ is not dominated by $g(n)$.

Function can be written as $n! = \Omega((2)^n)$

$f = \Omega(g)$

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

$T(n) = [T(n-2) + 2(n-1) - 1] + 2n - 1$

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

$\cancel{T(n) =}\ T(n-3) = T(n-4) + 2(n-3) - 1$

$T(n) = T(n-4) + 2(n-3) + 2(n-2) + 2(n-1) + 2n - 4$

$T(n-4) = T(n-5) + 2(n-4) - 1$

$T(n) = T(n-5) + 2(n-4) + 2(n-3) + 2(n-2)$
$+ 2(n-1) + 2n - 5$

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

$\because k = n$

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

$= 0 + 2 + 4 + 6 + \ldots + 2n - n$

$= \frac{\cancel{2}n(n+1)}{\cancel{2}} - n$

$= n^2 + n - 1$

$= n^2$

$= O(n^2)$

b) $T(n) = 9T\left(\frac{n}{3}\right) + n^2 \log n$

$T(n) = aT\left(\frac{n}{b}\right) + O(n^d)$

$T(n) = aT\left(\frac{n}{b}\right) + f(n)$

$T(n) = 9T\left(\frac{n}{3}\right) + n^2 \log n$

$a = 9$

$b = 2$

$f(n) = n^d \log^k n$

$f(n) = O(n^2 \log n)$

$d = 2$

$9 > 2^2$

$9 > 4$

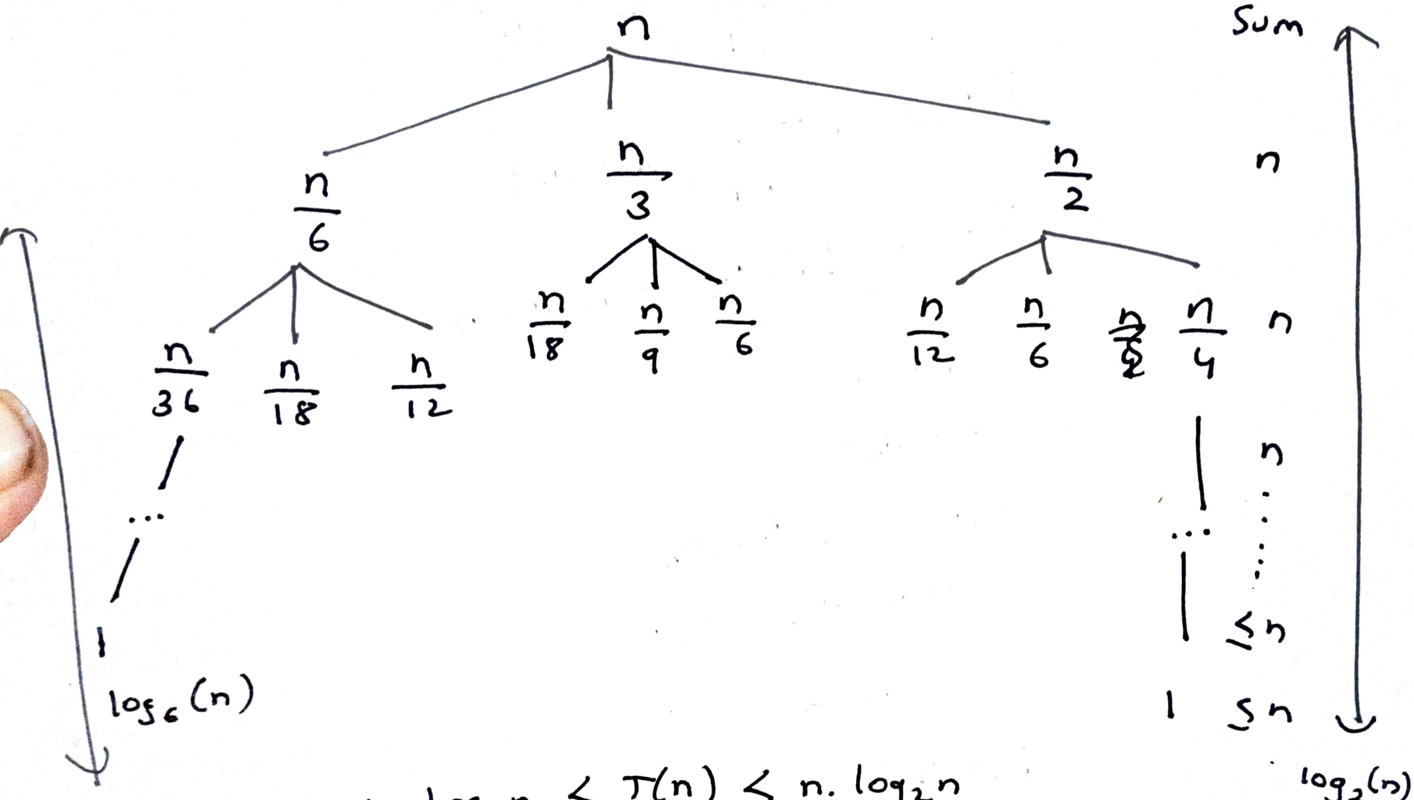
if $a > b^d$

$T(n) = O(n^{\log_b a})$

$= O(n^{\log_2 9})$

$= O(3n^{3.17})$

c) $T(n) = T\left(\frac{n}{2}\right) + T\left(\frac{n}{3}\right) + T\left(\frac{n}{6}\right) + n$

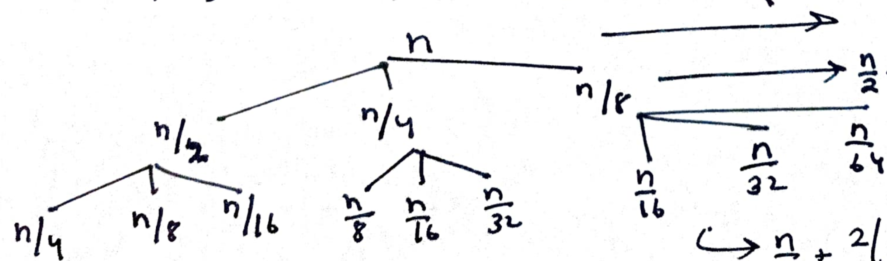


$$n \cdot \log_6 n < T(n) < n \cdot \log_2 n$$

$$T(n) = \Theta(n \cdot \log n)$$

d) $T(n) = \sum_{i=1}^{\log_2 n} T\left(\frac{n}{2^i}\right) + n$

$T(n) = T\left(\frac{n}{2}\right) + T\left(\frac{n}{4}\right) + T\left(\frac{n}{8}\right) + \cancel{T\left(\frac{n}{16}\right)} + n$



n

$\frac{n}{2} + \frac{n}{4} + \frac{n}{8} = \frac{7n}{8}$

$\hookrightarrow \frac{n}{4} + 2\left(\frac{n}{8}\right) + 2\left(\frac{n}{16}\right) + 2\left(\frac{n}{32}\right) + \frac{n}{64}$

$= \frac{49n}{64} = \left(\frac{7}{8}\right)^2 n$

~~To f~~ Similarly, next step count $= \left(\frac{7}{8}\right)^3 n$

$\vdots$

$\left(\frac{7}{8}\right)^k n$

To find k

$$\frac{n}{2^{\cancel{k}i}} = 1 \Rightarrow \frac{n}{2^k} = 1$$

$$k = \log_2 n$$

$$T(n) = n\left[1 + \left(\frac{7}{8}\right)^1 + \left(\frac{7}{8}\right)^2 + \cdots \left(\frac{7}{8}\right)^{\log_2 n}\right]$$

$$= n$$

$$T(n) = O(n)$$

4.a) int MaxRepeating (int arr[], int n int k)

```
{
// Iterate through input array, for every element
  arr[i], increment arr[arr[i] % k] by k
   for i → 0 to n
      arr[arr[i] % k] += k;  end for;
 // Find index of the maximum repeating element
   int max = arr[0], result = 0;
  for i → 1 to n
    if (arr[i] > max)
        max = arr[i];
        result = i;
    end for
// Uncomment this code to get original array back
   for i → 0 to n
   arr[i] → arr[i] % k;
// Return index of maximum element
  print result;

merge (int a[], int p, int q, int r)
  {
     n1 ← q-p+1
     n2 ← r-p;
   for i → 1 to n1
     do L[i] → A[p+i-1]  // copy data to temp arrays
   for j → 1 to n2
     do R[j] → A[q+j];
    L[n1+1] → NULL;
    L[n2+1] → NULL;
     i → 1, j → 1;
   for i → p to r
   do if L[i] ≤ R[j] then A[k] → L[i]
                     i++;
       else  A[k] → R[j]   /* merge temp arrays back
                j++;              into a
```

```
mergesort (A, p, r)
    if p<r
    then q ← (p+r)/2          // Recursion function
    mergesort (A, p, q)       // sort first & second halves
    mergesort (A, q+1, r)
    merge (A, p, q, r)

void PrintRandoms (int arr[], int lower, int upper, int count)
{
    for loop → 0, to count
        num = (rand() % (upper - lower + 1)) + lower;
        print num;
        arr[i] → num;
}
int main()
{
    read n
    read arr[n]
    printRandoms (arr, 0, n-1, n);
    maxRepeating (arr, n, n);
}
```

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

$T(n) = \sum_{k=1}^{\log n} 2^k c = c\left(2^{\log n+1} - 2\right)$

$T(n) = \log n + 2\log\left(\frac{n}{2}\right) + 4\log\left(\frac{n}{4}\right) + \ldots$

$\sum_{k=1}^{m} 2^k \log\left(\frac{n}{2^k}\right) = \sum_{k=1}^{m} 2^k(\log n - k) = \log n \sum_{k=1}^{m} 2^k - \sum_{k=1}^{m} k 2^k$

$= \log n\left(2^{m+1} - 2\right) - \left(m2^{m+1} - 2^{m+1} + 2\right)$

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

$= 2n - \log_2 n - 2$

$= O(n\log n)$

maxRepeat Function is used to find maximum repeating element in array of random element produced by random.

Iterate through array for every element arr[] increment arr[arr[i] % k] by k.

Find max value in modified array. Index of the max value is the maximum repeating element.

If we want to get original array back we can ~~it~~ traverse through array and do arr[i] = arr[i] % k where i is from 0 to n-1.

Since we use arr[i] % k as index and add value k at index arr[i] % k, the index which is equal to maximum repeating element will have max value in the end.

5.a)

We take 2 recursive calls for memoization.

The user chooses ith card with value V_i. The opponent chooses jth coin. The opponent ~~tends to choose card~~ Rana chooses card which leaves player with minimum value.

We can collect value $V_i + (Sum - V_i) - F(i+1, j, sum - V_i)$ where Sum is sum of cards from index i to j.

The expression can be simplified to $Sum - F(i+1, j, sum - V_i)$.

$F(i,j)$ represents maximum value the user can collect from i to j cards.

arr[] is list of cards

$F(i,j)$ = maximum of $Sum - F(i+1, j, sum - arr[i])$, $Sum - F(i, j-1, Sum - arr[j])$)

Base case

$F(i,j) = max(arr[i], arr[j]$ if $j == i+1$)

~~e.g. 5, 3, 7, 10 5, 3, 7, 10~~

~~User collects maximum value 15 → (10 + 5)~~

~~User takes 5.~~

~~Rana takes 3.~~

e.g. 8, 15, 3, 7

User takes 8

Rana takes 15

User takes 7

Rana takes 3

Total value for user = 15 (8+7)

By analyzing recursive forms

Equation : $T(n) = T(n-1) + n$

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

$T(n-2) = T(n-3) + n-2$

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

$T(n) = T(n-3) + n-2 + n-1 + n$

$T(n) = T(n-k) + kn - \frac{k(k-1)}{2}$.

Base case

$n-k = 1$

$T(1)$

$k = n-1$

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

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

$= O(n2)$