Design and Analysis of Algorithm (DAA)

Practice Questions

[Set 1]

Dr. Dayal Kumar Behera

School of Computer Engineering
KIIT Deemed to be University, Bhubaneswar, India

Analysis

Let B(n), W(n) and A(n) denote the best case, worst case and average case running time of an algorithm respectively, executed on an input of size n.
Which of the following is always **TRUE**?

A. B(n) = O(W(n))
B. W(n) = Θ(A(n))
C. A(n) = O(B(n))
D. A(n) = Ω(W(n))
E. NONE

Ans: A. B(n) = O(W(n))

Analysis

Let B(n), W(n) and A(n) denote the best case, worst case and average case running time of an algorithm respectively, executed on an input of size n.
Which of the following is always **TRUE**?

A. $B(n) = \Theta(W(n))$
B. $W(n) = \Omega(A(n))$
C. $A(n) = O(B(n))$
D. $B(n) = \Omega(W(n))$
E. NONE

Ans: B. $W(n) = \Omega(A(n))$

Analysis

Let B(n), W(n) and A(n) denote the best case, worst case and average case running time of an algorithm respectively, executed on an input of size n. Which of the following is **NOT always TRUE**?

A. B(n) = O(W(n))

B. W(n) = Ω(A(n))

C. A(n) = Ω(B(n))

D. B(n) = Ω(W(n))

Ans: B(n) = Ω(W(n))

Analysis

Consider the following function.

```
int   fun(int n)
{
        if (n == 0 || n==1)
                return n;
        else
                return 1 + fun(n-1) + fun(n-1);
}
```

What is the least upper bound time complexity of the function fun?

A. O(n)

B. O(nlog n)

C. $O(n^2)$

D. $O(2^n)$

E. NONE

Ans: D

Analysis

Consider the following function.

```
int fun(int n, int A[]){
        int i, j, s=0;
        for(i = 1; i<= n; i++){
                j = n;
                while (j> 0){
                        s= s +j;
                        j = j - 2;
                }
        }
        return s;
}
```

What is the least upper bound time complexity of the function fun?

A. O(n)

B. O(nlog n)

C. $O(n^2)$

D. $O(2^n)$

E. NONE

Ans: C

Analysis

Consider the following code fragment.

```
int a[] = {2, 1, 4, 3, 5, 6, 7, 8, 9, 0}
int fun(int b[], int n)
{
        if (n==1)
                return b[n-1];
        else    return b[n] + 2*fun(b, n-1);
}
```

What is the result of the function call fun(a, 3) and What's the asymptotic running time of this function in terms of n respectively.

A. 18, $\Theta(n)$

B. 18, $\Theta(2^n)$

C. 19, $\Theta(n)$

D. 19, $\Theta(2^n)$

E. NONE

Ans: C

Analysis

Consider the following code fragment.

```
int a[] = {2, 1, 4, 3, 5, 6, 7, 8, 9, 0}
int fun(int b[], int n)
{
        if (n==1)
                return b[n-1];
        else    return b[n] + fun(b, n-1) + fun(b, n-1);
}
```

What is the result of the function call fun(a, 3) and What's the asymptotic running time of this function in terms of n respectively.

A. 18, $\Theta(n)$

B. 18, $\Theta(2^n)$

C. 19, $\Theta(n)$

D. 19, $\Theta(2^n)$

E. NONE

Ans: D

Sorting

If all the elements in an input array are same, for example {4,4,4,4,4,4}, Which of the following sorting algorithm has the lowest time complexity?

A. Insertion Sort
B. Quick Sort
C. Merge Sort
D. Both Quick & Merge Sort
E. NONE

Ans: A

Sorting

If all the elements in an input array are same, for example {4,4,4,4,4,4}, Which of the following sorting algorithm has the highest time complexity?

A. Insertion Sort
B. Quick Sort
C. Merge Sort
D. Both Quick & Merge Sort
E. NONE

Ans: B

Sorting

Match the following pairs:

P. O(log n)	i. Worst case Quick Sort
Q. O(n)	ii. Binary Search
R. O(n log n)	iii. Best Case Insertion Sort
S. O(n2)	iv. Merge Sort
	v. Linear Search

A. P-ii, Q-iii, Q-v, R-iv, S-i,
B. P-iii, Q-ii, R-iv, R-i, S-i
C. P-i, Q-ii, R-iv, S-iii, S-i
D. P-iv, P-v, Q-ii, R-i, S-iii
E. NONE OF THE OPTION

Ans: A

Sorting

Match the following pairs:

P. O(1)	i. Best case Insertion Sort
Q. O(n)	ii. Best case Linear Search
R. O(n log n)	iii.Worst case Bubble Sort
S. O(n2)	iv. Heap Sort
	v.Merge Sort

A. P-ii, Q-iii, R-iv, S-i, S-iii
B. P-iii, Q-ii, Q-v, R-iv, S-i
C. P-ii, Q-i, R-iv, R-v, S-iii
D. P-iv, Q-ii, Q-v, R-i, S-iii
E. NONE OF THE OPTION

Ans: C

Sorting

In an array of n integers first n/2 elements are sorted in ascending order, rest sorted in descending order. What is the minimum time required to sort the data in ascending order?

A. O(log n)

B. O(nlog n)

C. O(n)

D. $O(n^2)$

E. NONE

Ans: C

Heap

What is the minimum time required to merge two max-heaps, each having n elements, into one max heap?

A. O(1)

B. O(log n)

C. O(nlog n)

D. O(n)

E. NONE

Ans: D

Heap

Suppose we are sorting an array of eight integers using heapsort, and we have just finished some heapify (either maxheapify or minheapify) operations. The array now looks like this: 16 14 15 10 12 27 28. How many heapify operations have been performed on root of heap?

```
//As one element got its position out of eight
1.    BUILD-MAX-HEAP(A)                          //one
2.    for i ← length[A] downto 2
3.        do exchange A[1] ↔ A[i]
4.            Heap-Size(A) = Heap-Size(A) -1
5.            MAX-HEAPIFY(A, 1)                  //two
```

Ans: 2 times

Heap

Let Array A[1..11]={9, 5, 7, 3, 2, 6, 7, 3, 1, 2, 1} is a max-heap. What will be resultant max-heap, if the value at index 5 is changed to 7.

A. {9, 7, 7, 3, 5, 6, 7, 3, 1, 2, 1}

B. {9, 5, 7, 3, 3, 6, 7, 3, 1, 2, 1}

C. {9, 7, 7, 3, 5, 6, 7, 3, 1, 2, 2}

D. {9, 7, 7, 5, 2, 6, 7, 3, 1, 2, 1}

E. NONE

Ans: A

Heap

Let Array A[1..11]={9, 5, 7, 3, 2, 6, 7, 3, 1, 2, 1} is a max-heap. What will be resultant max-heap, if the value at index 5 is changed to 3.

A. {9, 7, 7, 3, 5, 6, 7, 3, 1, 2, 1}

B. {9, 5, 7, 3, 3, 6, 7, 3, 1, 2, 1}

C. {9, 7, 7, 3, 5, 6, 7, 3, 1, 2, 2}

D. {9, 7, 7, 5, 2, 6, 7, 3, 1, 2, 1}

E. NONE

Ans: B

Heap

Let Array A[1..11]={9, 5, 7, 3, 2, 6, 7, 3, 1, 2, 1} is a max-heap. What will be resultant max-heap, if the value at index 11 is changed to 7.

A. {9, 7, 7, 3, 5, 6, 7, 3, 1, 2, 1}

B. {9, 5, 7, 3, 3, 6, 7, 3, 1, 2, 1}

C. {9, 7, 7, 3, 5, 6, 7, 3, 1, 2, 2}

D. {9, 7, 7, 5, 2, 6, 7, 3, 1, 2, 1}

E. NONE

Ans: C

Quick-Sort

Find the worst case time complexity of quick sort and its recurrence.

a) Time complexity is O(n^2) and recurrence is T(n) = T(n-2) + O(n)

b) Time complexity is O(n^2) and recurrence is T(n) = T(n-1) + O(n)

c) Time complexity is O(nLogn) and recurrence is T(n) = 2T(n/2)

d) Time complexity is O(nLogn) and recurrence is T(n) = T(n/10) + T(9n/10) + O(n)

Ans: a and b

Recurrence

Solve the following recurrence relation?

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

a = 7, b = 2, and f(n) = 3n^2 + 2

So, f(n) = O(n^c), where c = 2.

logb(a) = log2(7) = 2.81 > 2

It follows from the first case of the master theorem that

T(n) = θ(n^2.8) and implies O(n^2.8) as well as O(n^3)

Recurrences

Solve the following recurrences-

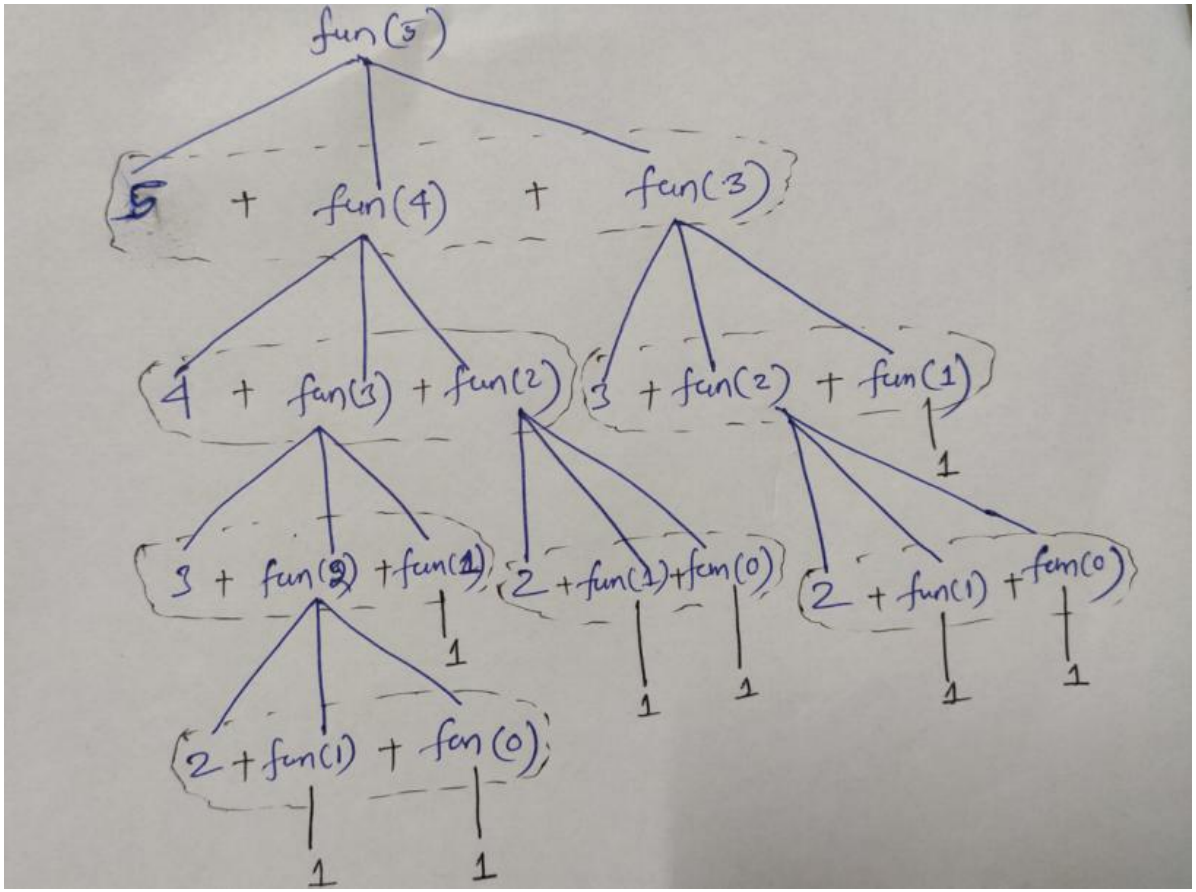
I. $T(n) = \sqrt{2}\,T\left(\frac{n}{2}\right) + \log n$ $\Theta(\sqrt{n})$

II. $T(n) = T\left(\frac{n}{4}\right) + T\left(\frac{3n}{4}\right) + cn$ O (nlgn)

Recurrence

```
int fun( int n)
{
    if (n<=1)
        return 1;
    else
        return n + fun(n-1) + fun(n-2);
}
```

How many additions are performed to compute fun(5)?



Number of major function calls where two additions are performed: 7

Number of addition in each function call: 2

So total number of additions=7x2=14

Recurrence

```
int fun( int n)
{
    if (n<=1)
        return 1;
    else
        return n + fun(n-1) + fun(n-2);
}
```

Assuming that each addition/subtraction takes constant time, write a recurrence relation for the running time of fun(n) & solve the recurrence.

Solving Recurrence

$T(n) = T(n-1) + T(n-2) + 1$ ……(1)

Now the recurrence is solved as follows:

Lower Bound:
Establishing a lower bound by approximating that $T(n-1) \sim T(n-2)$,
As $T(n-1) \geq T(n-2)$, hence lower bound, eq-1 becomes
$T(n) = T(n-2) + T(n-2) + 1$

Upper Bound:
Establishing an upper bound by approximating that $T(n-2) \sim T(n-1)$,
As $T(n-1) \geq T(n-2)$, hence upper bound, the recurrence eq-1 becomes
$T(n) = T(n-1) + T(n-1) + 1$

Solving Recurrence

$T(n) = T(n-1) + T(n-2) + 1$ ……(1)

Lower Bound Solution:

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

$= 2T(n-2) + 1$
$= 2\{2T(n-4) + 1\} + 1$
$= 2^2T(n-4) + 3$
$= 2^2\{2T(n-6) + 1\} + 3$
$= 2^3T(n-6) + 7$
$= 2^3\{2T(n-8) + 1\} + 7$
$= 2^4T(n-8) + 15$
……
$= 2^kT(n-2*k) + 2^k-1$

To find out the value of k for which: $n - 2k = 0 \Rightarrow k = n/2$

$= 2^{n/2}T(0) + 2^{n/2}-1$
$= 2^{n/2} *1 + 2^{n/2}-1$
$= 2^{n/2} *1 + 2^{n/2}-1$
$\sim O(2^{n/2})$

Solving Recurrence

$T(n) = T(n-1) + T(n-2) + 1$ ……(1)

Upper Bound Solution:

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

$= 2T(n-1) + 1$
$= 2\{2T(n-2) + 1\} + 1$
$= 2^2T(n-2) + 3$
$= 2^2\{2T(n-3) + 1\} + 3$
$= 2^3T(n-3) + 7$
$= 2^3\{2T(n-4) + 1\} + 7$
$= 2^4T(n-4) + 15$

……
$= 2^kT(n-k) + 2^k-1$

To find out the value of k for which: $n - k = 0 \Rightarrow k = n$

$= 2^nT(0) + 2^n-1$
$= 2^n * 1 + 2^n-1$
$= 2^n * 1 + 2^n-1$
$\sim O(2^n)$

Hence, **the time complexity of function fun() in worst case = $O(2^n)$**

Merge-Sort [2021]

a) Write the MERGE-SORT(A, p, r) procedure where at each step it divides the array/sub-array into two parts such that second part contains elements twice of first part instead of dividing at middle.

b) For array A={10, 45, 15, 40, 10, 20, 40, 25, 35}, MERGE-SORT(A, 1, 9) is applied to sort the array in ascending order. Show in diagram how this procedure is applied to this array.

Merge-Sort [2021]

a) MERGE-SORT procedure

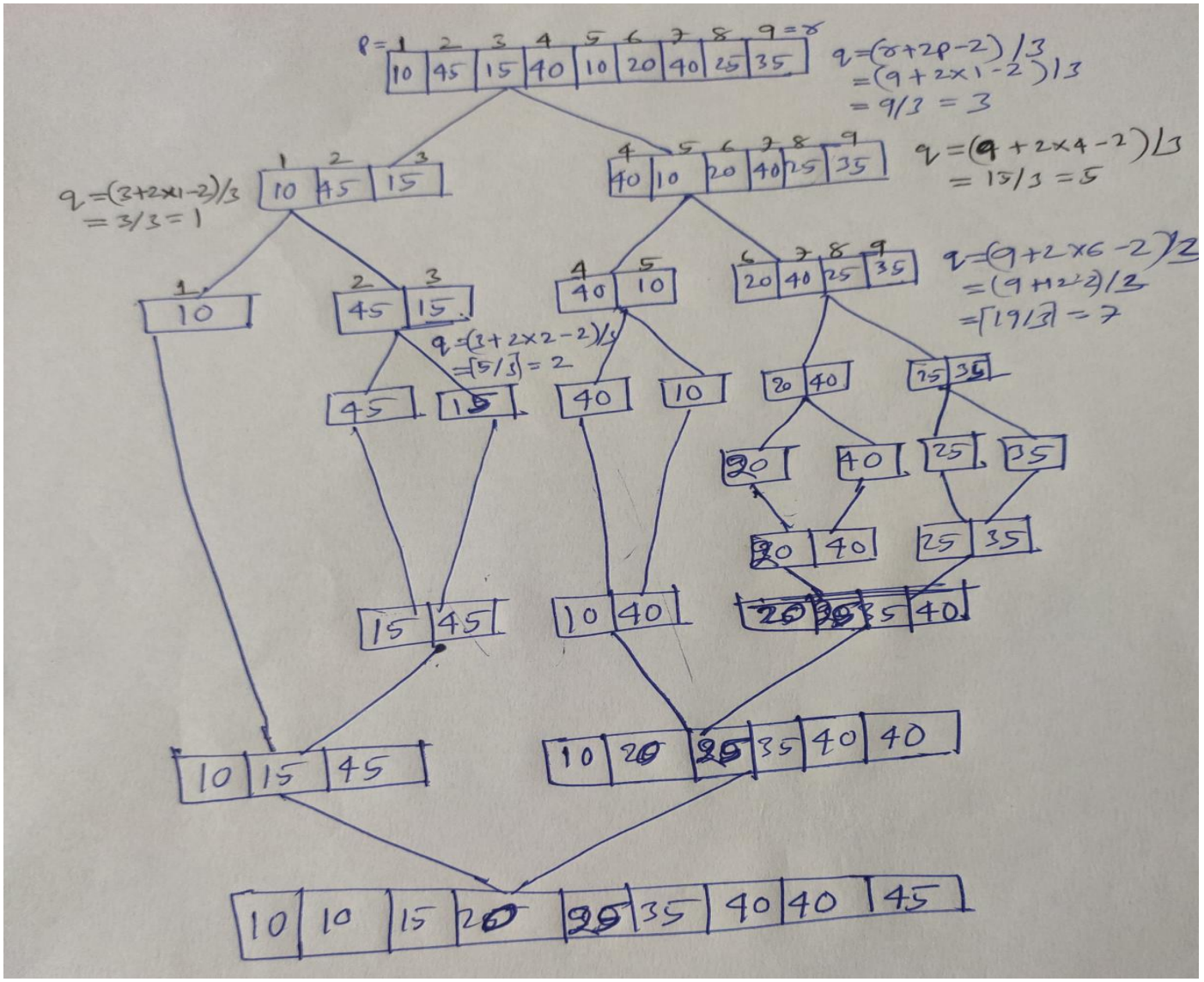
```
//merge sort is applied to the array A to sort the array in ascending order from the
//lower bound/index p to upper bound/index r.
MERGE-SORT (A, p, r)
{
      if (p<r)
      {
            //divides the the array/sub-array into two parts such that second part
            //contains elements twice of first part
            q←(r+2p-2)/3;
            MERGE-SORT (A, p, q);
            MERGE-SORT (A, q+1, r);
            MERGE (A, p, q, r);
      }
}
```

MERGE: No change in the algorithm discussed.

Merge-Sort [2021]

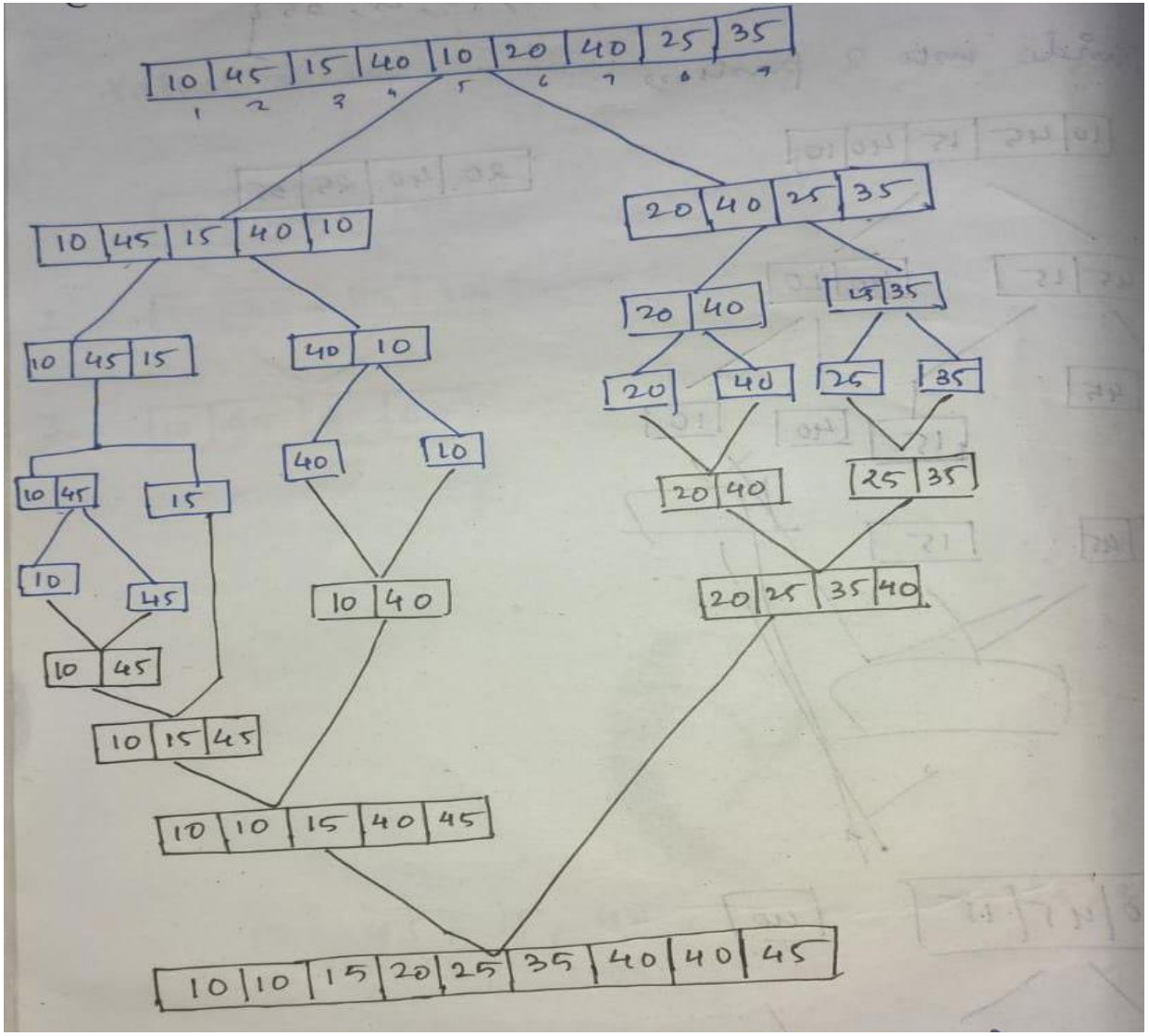
Taking the division point

q=(r+2p-2)/3



Taking the division point

q=(p+r)/2




Each of your
actions will
have an
impact on your
future.


Once you know
who is walking
with you on your path.
you will never
be afraid.

Thank you