Q. 5.

mergesort(arr[], l, r)

⇒ arr[] is the array to be sorted

⇒ l is the leftmost element

⇒ r is the rightmost element

Algorithm

→ Find the middle and divide the array into two halves.

middle = (l+r)/2

→ Call mergesort() for first half

mergesort(arr, l, middle)

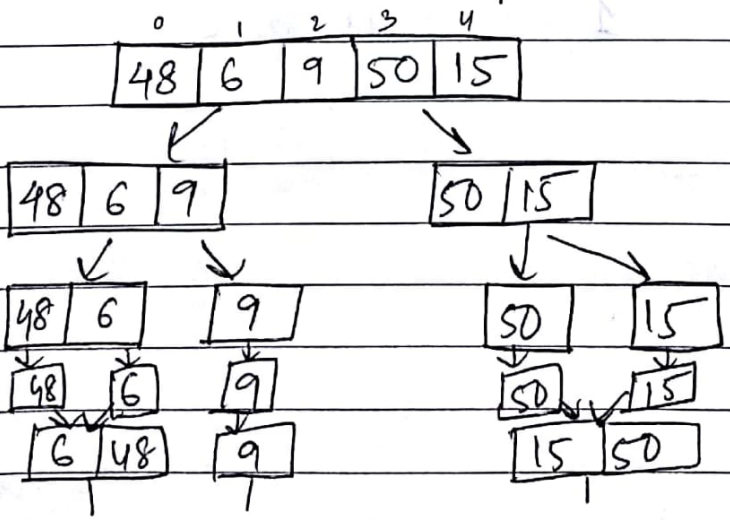
→ Call mergesort for second half

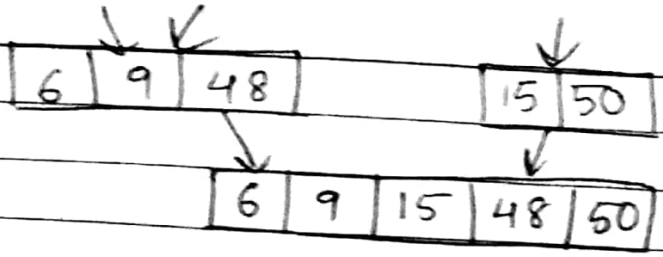
mergesort(arr, middle+1, r)

→ Merge the two halves by calling

merge(arr, l, middle, r)

Let's take an example





For this case the DAG will be as follows :-

(arr, 0, 2) → (arr, 0, 1) → (arr, 0, 0)

{18, 6, 9} R

{48, 6, 9} R
{48, 50} W

{18, 50} R
{48} W
{6} W

~~(arr, 3, 4)~~

(arr, 3, 3)
{50, 15} R
{50} W

(arr, 1, 1)
{50} R

(arr, 2, 2)
{9} R

(arr, 3, 4) ⇐
{50, 15} R

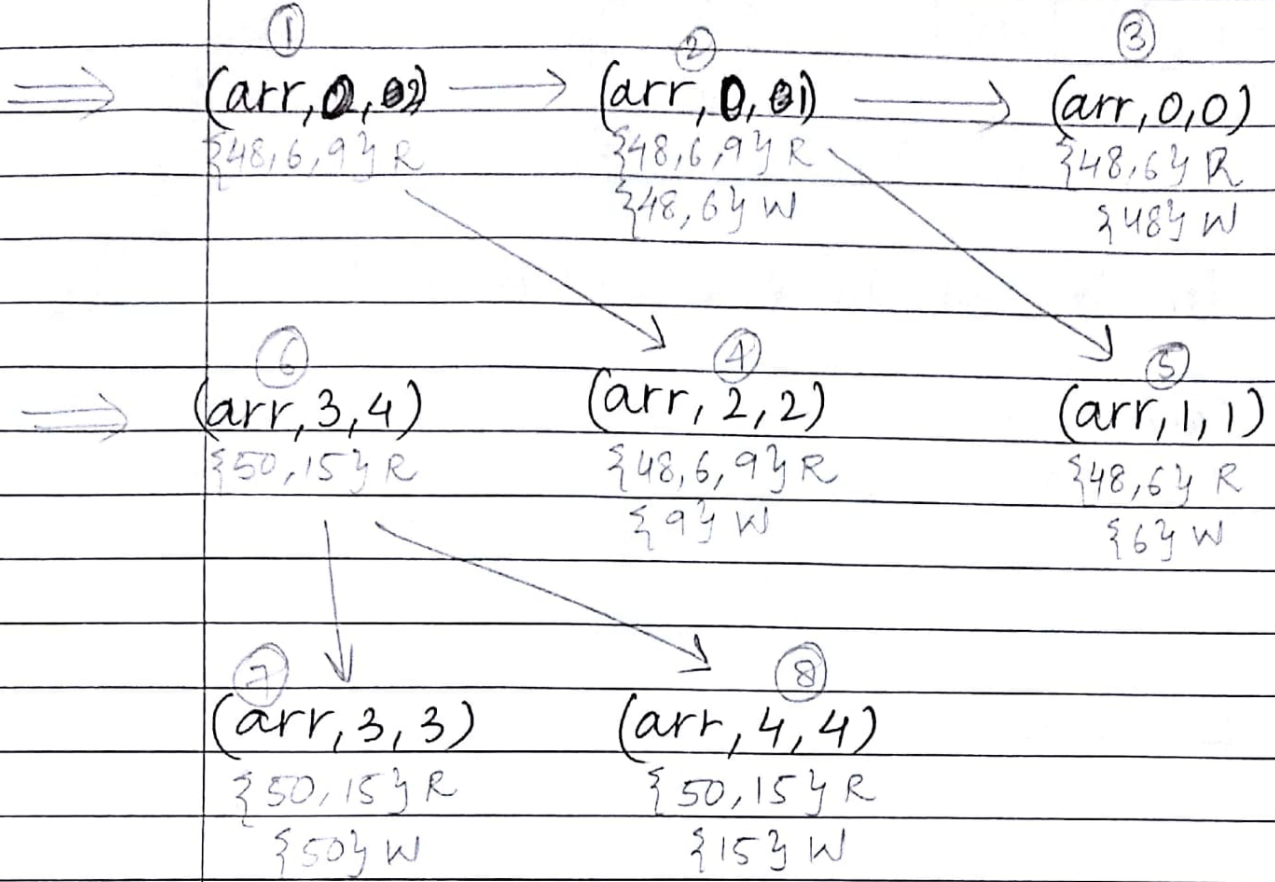
* All the tasks will not have the same processing time.

(arr, 4, 4)
{50, 15} R
{15} W

CRITICAL PATH :- 4

WIDTH :- 4

WORK :- 4



WIDTH :- 4 = N-1

CRITICAL PATH :- 3 = N-2

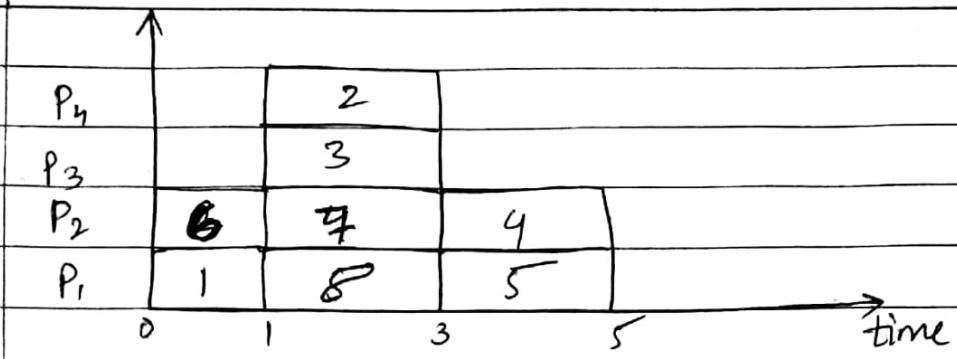
WORK :- $N - \frac{1}{2} - \frac{1}{2} = N-1 = 5-1 = 4$

All the tasks will not have the same processing time.

Tasks (arr,0,2) and (arr,3,4) will have lesser processing time as compared to other tasks as these two tasks only read data.

The other tasks will have equal processing time.

__/__/____



Assuming one read takes 1 unit of time and one write takes one unit of time.

For a case of maximum possible parallelism, we can consider N/2 number of processors if N is the no. of elements that needed to be sorted.

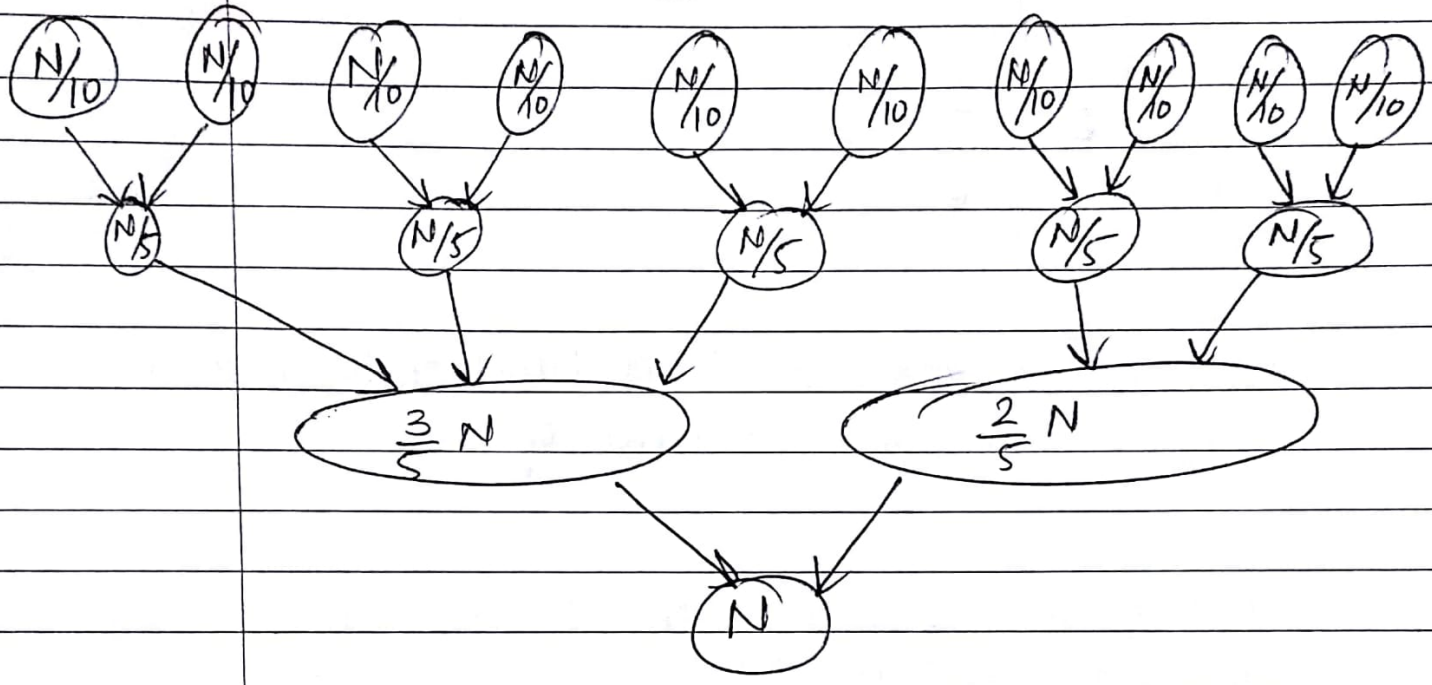
~~For the example considered earlier we~~
~~N/2 = 8/2~~

Consider a case of 10 elements to be sorted.
∴ N/2 = 10/2 = 5 processors can be used.

To use 5 processors we would need to change the algorithm a little.
Instead of ~~the~~ splitting the array into two halves in the first step, split each element as a singular node. Then club two nodes by sorting and then four nodes and so on.

__/__/____

* For N elements in the array.



In this case, we can use a maximum of 5 processors at the N/5 node stage.

MERGE SORT

The merge sort algorithm divides the n-element sequence to be sorted into two subsequences of n/2 elements each and then combine the sorted subsequences.

The following is the merge sort algorithm.

```
merge(A, p, q, r)
 n1 = q - p + 1
 n2 = r - q
 Let A1[1 .. n1+1] and A2[1 .. n2+1] be new arrays.
 for i = 1 to n1
      A1[i] = A[p + i - 1]
 for j = 1 to n2
      A2[j] = A[q + j]
 A1[n1 + 1] = ∞
 A2[n2 + 1] = ∞
 i = 1
 j = 1
 for k = p to r
   if A1[i] ≤ A2[j]
      A[k] = A1[i]
      i = i + 1
   else A[k] = A2[j]
      j = j + 1
```

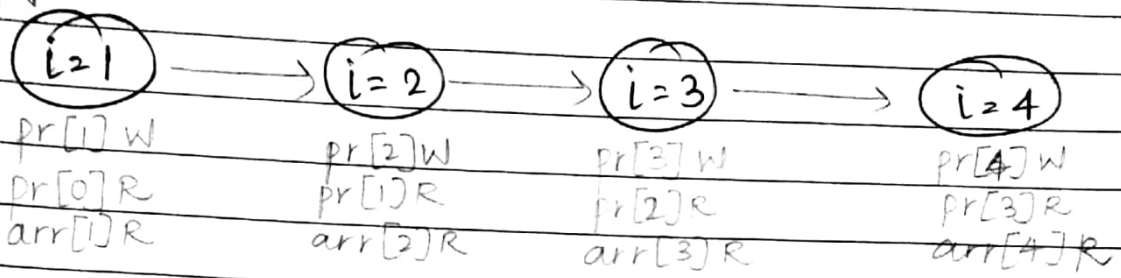
Here, A is the array to be sorted, p, q and r are the indices into the array such that $p \leq q < r$.

Q-4

__/__/__

```
void prefixsum(int* arr, int n, int* pr){
    pr[0] = arr[0];
    for(int i=1; i<n; ++i)
        pr[i] = pr[i-1] + arr[i];
}
```

pr[0] = arr[0]



WIDTH :- 1

WORK :- (N-1)1 = (N-1)

CRITICAL PATH :- N

Algo! * To make it parallel, we use an algorithm that has shorter span and more parallelism.

Hillis & Steele algorithm is given as,

```
for(i=0; i<[log2n] - 1; i++)
    for(j=0; j<=n-1; j++)
        if j < 2^(i+1)
            x_j^(i+1) = x_j^i
        x_j^(i+1) = x_j^i + x_(j-2^i)^i
```

x_j^i means value of jth element of array x in timestep i.

*Algo 2 Another approach is a work-efficient parallel prefix sum algorithm.

1. Starting from index 0, ~~club two~~ make pairs of 2 and compute their sum

$\therefore z_0 = x_0 + x_1$
$z_1 = x_2 + x_3$
$z_3 = x_4 + x_5$ etc

2. Recursively compute the prefix sum $w_0, w_1, w_2, \ldots$ of the sequence $z_0, z_1, z_2, \ldots$

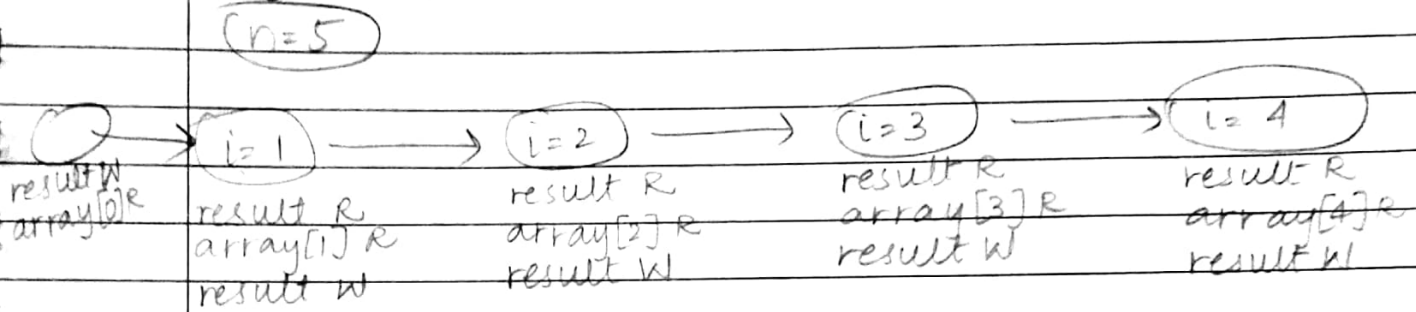
3. The final sequence will be $y_0, y_1, y_2, y_3 \ldots$ which can be computed as follows :-
$y_0 = x_0$
$y_1 = z_0$
$y_2 = x_2 + z_0$
$y_3 = z_0 + z_1 = w_0$
$y_4 = x_4 + w_0$ etc.

work is 2 times the work of a sequential algorithm.

__/__/____

Q-3 Considering int and sum case

```
int reduce(int * array, size_t n){
    int result = array[0];
    for(int i=1; i<n; ++i)
        result = sum(result, array[i]);
    return result;
}
```



WORK :- N

WIDTH :- 1

CP :- N-1

__/__/____

* Rewriting the code to introduce one local variable per processor to store partial computation.

For P processors, we can divide the given array into P sub arrays and compute ~~partial~~ sums of these sub arrays. Combine all the sums to get the final result.

consider p = 4

```
int reduce(int * array, size_t n){
  ~~int p1=~~
  for(int i=0; i< n/4 ; ++i){
      ~~int p1 = 0;~~
      ~~int p~~
```

consider 4 processors

```
int reduce(int* array, size_t n){
    int p1=0, p2=0, p3=0, p4=0, sum=0;
    for(int i=0; i< n/4 ; ++i){
        p1 = sum(p1, array[i]);
    }
    for(int i= n/4; i< n/2; ++i){
        p2 = sum(p2, array[i]);
    }
    for(int i= n/2; i< 3n/4 ; ++i){
        p3 = sum(p3, array[i]);
    }
    for(int i= 3n/4; i< n ; ++i){
        p4 = sum(p4, array[i]);
    }
```

__/__/____

sum2 = P1 + P2 + P3 + P4 ;
return sum ;
}

consider n = 8

i=0 → i=1 → i=2 → i=3

i=0: array[0] R, P1 R, P1 W
i=1: array[1] R, P1 R, P1 W
i=2: array[2] R, P2 R, P2 W
i=3: array[3] R, P2 R, P2 W

i=4 → i=5 → i=6 → i=7

i=4: array[4] R, P3 R, P3 W
i=5: array[5] R, P3 R, P3 W
i=6: array[6] R, P4 R, P4 W
i=7: array[7] R, P4 R, P4 W

WORK :- $\frac{n}{4} * \frac{n}{4} * \frac{n}{4} * \frac{n}{4} = \frac{n^4}{256} = \frac{8 \times 8 \times 8 \times 8}{256} = 16$

WIDTH :- $\frac{n}{2} = \frac{8}{2} = 4$

CP :- $\frac{n}{4} = \frac{8}{4} = 2$

P4	i=6	i=7
P3	i=4	i=5
P2	i=2	i=3
P1	i=0	i=1

0 t 2t time

Assuming each task takes t units of time.

3.2 Variants

1. The parallel version would be correct for int, max. We could calculate individual maximums for the subarrays and then obtain the total maximum by max of all individual max.

2. I think the parallel version will be correct for string, concat as we can concatenate substrings and then sequentially concatenate those substring concatenations.

3. & 4. float, sum and float, max should also be compatible with the parallel version of the code as the individual sum and max calculated will be in float and also, the final sum and max have float as their datatype which matches the returntype of the function.

* References:

Algo 1 and Algo 2 of Q-4 are from Wikipedia