Question: Assuming you have P processors, rewrite the code to introduce one local variable per processor to store partial computation so as to achieve more parallelism. What is the width, critical path and work ?
Ans:

Q.1)

1.1 Program

```
template <typename T, typename op>
    T reduce (T* array, size_t n)
        T result = array [0]        # Task 1
        for (int i=1; i<n; ++i)
            result = op (result, array [i]); #Task 2
        return result                        # Task3
```

Let say N=4 and array = [1, 2, 3, 4]

① result = 1 (reduce_pro)
↓ int i=1
② result = op(result, array[1])
= op(1,2) # since op is sum
= 3
[reduce_result_array[1])
↓ result
③ return result
return 3 (reduce_post)

④ result = 3 (reduce_pre)
↓ i=2
⑤ result = op(result, array[2])
= op(3,3)
= 6
(reduce_result, array[2])
↓ result
⑥ return result
return 6 (reduce_post)

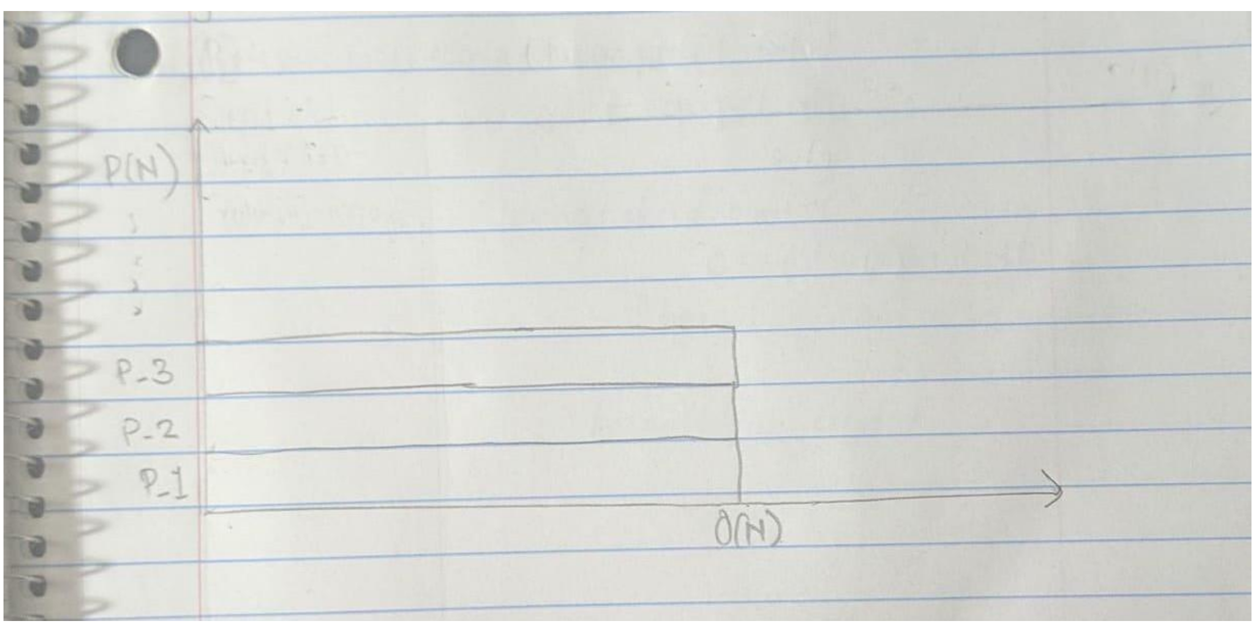
⑦ result = 6
(reduce_pre)
↓ i=3
⑧ result = op(result, array[3])
= op(6,4)
= 10
(reduce_result_array[3])
↓ result
⑨ return result
return 10 (reduce_post)

Since there is no parallelism, we will be changing the code as below:

```
Steps  template <typename T, typename op>
         T reduce (T* array, size_t n){
①          T value [P]        # Processor array to stored value
         for (int i=0; p<P; ++i){
②          value [i] = array[0]  # Storing inital value of array[0]
           for (int j=1; j<N j++j){              # in result
③            value [i]= op(value[i], array[j]); # Storing the value
                                  by suming value[i] and
                                  array[j] in value[i]
           }
         }
④        return value      # returning value
       }
```

P(N)

Question: What does a schedule look like on P processors?
Ans:



What is the width, critical path and work ?
Ans: Width should be theta(P),Critical Path theta(N) and Work O(P*N)

Question: Would that parallel version be correct for int, max? Why?
Ans: The parallel version will be correct for int,max. For each value in the value array which is int here, the program will run P times and replaces the max value each time. The max value will be decided between value[i] and array[j]. For each iteration once the max value is derived , it stores into the value array by returning value. Since each process is independent from each other there will be no dependencies. Hence the parallel version will be correct for int,max.

Question: Would that parallel version be correct for string, concat? Why?
Ans: The parallel version will be correct for string,concat. For each value in the value array which is string here, the program will run P times and replaces the concatenated value each time. The concatenated value will be obtained by joining the value at value[i] position and array[j] position. For each iteration once the concatenated value is derived , it stores that value in the value array. Since each process is independent from each other there will be no dependencies. Hence the parallel version will be correct for string,concat.

Question: Would that parallel version be correct for float, sum? Why?
Ans:The parallel version will be correct for float,sum. For each value in the value array which is float here, the program will run P times and replaces the summation value each time. The summation value will be obtained by summing the value at value[i] position and array[j] position. For each iteration once the summation value is derived , it stores that value in the value array. Since each process is independent from each other there will be no dependencies. Hence the parallel version will be correct for float,sum

Question: Would that parallel version be correct for float, max? Why?
Ans:The parallel version will be correct for float,max. For each value in the value array which is float here, the program will run P times and replaces the max value each time. The max value will be obtained by considering maximum value at value[i[position and array[j] position. For each iteration once the max value is derived , it stores that value in the value array. Since each process is independent from each other there will be no dependencies. Hence the parallel version will be correct for float,max.

prefixSum

Question: Rewrite this algorithm to make it parallel on P processors. (Hint: What goes wrong if you were blindly parallelising the code by cutting the work in say 3 chunks? You may have to add some work without changing the complexity in Big-Oh notation. A single pass on the array is not enough.)

Ans:

```
void prefixSum(int *arr, int n, int *pr){
	int lastNum;
	for(int height=0; height<=log(n-1); height++){
		for(int i=0; i<=n-1; i = 2^(d+1)){
			arr[i+2^(d+1)-1] = arr[i+2^(d)-1] + arr[i+2^(d+1)-1];
		}
	}
	lastNum = arr[n-1];
	arr[n-1] = 0;
	for(int height=log(n-1); height>=0; height--){
		for(int i=0; i<=n-1; i=2^(d+1)){
			int temp = arr[i+2^(d)-1];
			arr[i+2^(d)-1] = arr[i+2^(d+1)-1];
			arr[i+2^(d+1)-1] = temp + arr[i+2^(d+1)-1];
		}
	}
	for(int j=1;j<n;j++){
		pr[i] = arr[i];
	}
	pr[n-1] = lastNum;
}
```

Question: What is the work, width, and critical path of the algorithm you created?

Ans:

Work is **O(n)**.

Width is **n/2**. As in first and for loop we add left element value in right element which is like a binary tree. And in last for loop we copy each element from arr and store it in pr so there it can be (n-1).

Critical Path is **O(log n).**

1 Merge Sort

Ans:

```
Function mergeSort(Array, Length)
	If Length == 1 return Array
	//Parallel Sections
	//Start
	leftArray = mergeSort(0, Length/2) //Run on processor 0
	rightArray = mergeSort(Length/2 + 1, Length) // Run on processor 1
	//End
	Return merge(leftArray, rightArray)

Function merge(leftArray, rightArray)
	B = new Array
	While leftArray =! Empty AND rightArray != Empty
		If leftArray[0] < rightArray[0]
			B.insrtAtLast(leftArray[0])
			Remove leftArray[0]
		Else
			B.insrtAtLast(rightArray[0])
			Remove rightArray[0]
	While leftArray != Empty
		B.insrtAtLast(leftArray[eachElement])
	While rightArray != Empty
		B.insrtAtLast(rightArray[eachElement])
	Return B
```

Question: What are the work and critical path of the merge algorithm you created?

Ans:

Work is O(n) and critical path would be O(2*log n).

Question: If you used that parallel merge in merge sort, what would the work and critical path of merge sort become?

Ans:

Work will be the O(n). Critical path will be O(log n).