1 Transform

Consider the transform function:

```
void  transform (int* a, int* b, int n) {
      for (int i=0; i<n; ++i)
            b[i] = f(a[i]);
}
```

Question: What is the complexity of this function?
Answer : The Complexity of this function is 0(n)

Question: Extract the dependencies. Assume the call to `F` cost O(1). Assume calls to `f` are always independent. (Note: Yes this problem is VERY simple!)
Answer: Consider n=4, now for loop will run from 0 to 3. And b[i] only depends on a[i] value and this is one Task. And these 4 task can run independently so there so no dependency of other tasks.\

1	2	3	4
b[0] = f(a[0])	b[1] = f(a[1])	b[2] = f(a[2])	b[3] = f(a[3])

Question: What is the width?
Answer : The width will be the n as each task is independent.

Question:What is the work?
Answer : The work will be the sum of all n task's individual work.

Question:What is the critical path? What is its length?
Answer : Value of B can be dependent on a, however in this case we can consider b[i] = f(a[i]) as a single task and hence the critical path will be the task which is taking maximum processing time. Hence, the Critical path will be the task taking maximum processing time and the length will be the time of it.

2. Reduce

Consider the reduce function:

```
template <typename T, typename op >
      T reduce (T* array , size_t n) {
```

```
      T result = array [0];
      for (int i=1; i<n; ++i)
            result = op (result , array[i]);
      return  result;
}
```

Do not be scared by the syntax! In C++, templates allow you to replace types and values in a piece of code by a type or a value known at compilation time. This is similar to generics in Java.
So if you define
`T` as `int` and `op` as `sum` , it boils down to computing the sum of the array. You could use `op` as `Max` and compute the maximum value of the array. Assume `T` is `int` and `op` is `sum`.

Question: What is the complexity of this function?
Answer: The Complexity of this function is O(n)

Question: Extract the dependencies.
Answer: Consider n = 4, so the for loop will run from 1 to 3. First task in the function is assigning array[0] to result. And the value of the result is read and written in each task. So each task has dependency on previous result value.

T1 → T2 → T3 → T4

Question: What is the width?
Answer: The width will be 1 since there is only one independent task which is T result = array[0] and each task is dependent on previous task's value.

Question: What is the work?
Answer: The work will be the sum of all n task's individual work.

Question: What is the critical path? What is its length?
Answer: The critical path will start from the dependent task which is T result = array[0] and will end till the value of n. The length will be the sum of the processing time for T result = array[0] And sum of all the processing time for all the tasks till n.

3 Prefix sum

Prefix sum is an algorithm that has many uses in parallel computing. The algorithm computes pr[i] $=\sum_{j<i} arr[j], \forall\, 0 \le i \le n$ and is often written sequentially:

```
void  prefixsum (int* arr , int n, int* pr) {
      pr[0] = 0;
```

```
      for (int i=0; i<n; ++i)
            pr[i+1] = pr[i] + arr[i];
}
```

Question: What is the complexity of this function?
Answer: The complexity of of this function is O(n)

Question: Extract the dependencies.
Answer: Consider n = 4, so the for loop will run from 0 to 3. First task in the function is assigning 0 to pr[0]. And in for loop, in each task pr[i+1] uses the pr[i] value. So each task is dependent on previous value.

1 → 2 → 3 → 4
pr[0] → pr[1] → pe[2] → pr[3]

Question:What is the width?
Answer: The width will be 1 since there is only one independent task which is pr[0] = 0;

Question:What is the work?
Answer: The work will be sum till n

Question:What is the critical path? What is its length?
Answer: The critical path will start from the dependent task which is `pr[0] = 0;` and will end till the value of n. The length will be the sum of the processing time for `pr[0] = 0;` And sum of all the processing time for all the tasks till n .