Time and Space Trade-Offs

Example: Sorting by Counting

Definition:

In computer science, a **space-time** or **time-memory tradeoff** is a way of solving a problem in :

1.) Less time by using more memory) or,

2.) By solving a problem in very little space by spending a long time.

Types of Trade Off:

1. Compressed / Uncompressed Data
2. Re-Rendering / Stored Images
3. Smaller Code / Loop Unrolling
4. Lookup Table / Recalculation

Compressed / Uncompressed Data

- A space -time trade off can be applied to the problem of data storage.
- If data is stored uncompressed,it takes more space but less time.
- If the data is stored compressed, it takes less space but more time to run the decompression algorithm.

Re-Rendering / Stored Images

- Storing only the source and rendering it as an image everytime the page is requested would be trading time for space.

  More time used but less space.

- Storing the images would be trading space for time.

  More space used but less time.

Smaller Code(with loop) / Larger Code (without loop)

- Smaller code occupies **less space** in memory but it requires **high computation time** which is required for jumping back to the beginning of the loop at the end of each iteration.

- Larger code or loop unrolling can be traded for higher program speed. It occupies **more space** in memory but requires **less computation time**.

  (as we need not perform jumps back and forth since there is no loop.)

Lookup Table / Recalculation

1. In lookup table, an implementation can include the entire table which **reduces computing time** but **increases** the amount of **memory needed.**

1. It can recalculate i.e. compute table entries as needed, **increasing computing time** but **reducing memory requirements**.

Example

More time, Less space	More Space, Less time
1. int a,b; 2. printf("enter value of a \n"); 3. scanf("%d",&a); 4. printf("enter value of b \n"); 5. scanf("%d",&b); 6. b=a+b; 7. printf("output is:%d",b);	1. int a,b,c; 2. printf("enter values of a,b and c \n"); 3. scanf("%d%d%d",&a,&b,&c); 4. printf("output is:%d",c=a+b);

Counting sort

- Step-1: Create a count array to store the count of each unique object.
- Step-2: Modify count array by adding the previous number.
- Step-3: Create output array by decrese count array.

Let the Array in range 0 to 5

Input	1	4	3	2	3	5	2

Create an array that will hold the count of each number, with index ranges from 0 to 5

	[0]	[1]	[2]	[3]	[4]	[5]
Count	0	1	2	2	1	1

Modify count array by adding the previous number:

Input	1	4	3	2	3	5	2

	[0]	[1]	[2]	[3]	[4]	[5]
Sum Count	0	1	3	5	6	7

Output each object from the input sequence followed by decreasing count by 1:

Input	1	4	3	2	3	5	2

	[0]	[1]	[2]	[3]	[4]	[5]
Sum Count	0	1	3	5	6	7

	[1]	[2]	[3]	[4]	[5]	[6]	[7]
output							

Output each object from the input sequence followed by decreasing count by 1:

Input	1	4	3	2	3	5	2

	[0]	[1]	[2]	[3]	[4]	[5]
Sum Count	0	1	3	5	1	1

	[1]	[2]	[3]	[4]	[5]	[6]	[7]
output	1						

Input	1	4	3	2	3	5	2

	[0]	[1]	[2]	[3]	[4]	[5]
Sum Count	0	0	3	5	6	7

	[1]	[2]	[3]	[4]	[5]	[6]	[7]
output	1					4	

Input	1	4	3	2	3	5	2

	[0]	[1]	[2]	[3]	[4]	[5]
Sum Count	0	0	3	5	5	7

	[1]	[2]	[3]	[4]	[5]	[6]	[7]
output	1				3	4	

Input	1	4	3	2	3	5	2

	[0]	[1]	[2]	[3]	[4]	[5]
Sum Count	0	0	3	4	5	7

	[1]	[2]	[3]	[4]	[5]	[6]	[7]
output	1		2		3	4	

Input	1	4	3	2	3	5	2

	[0]	[1]	[2]	[3]	[4]	[5]
Sum Count	0	0	2	4	5	7

	[1]	[2]	[3]	[4]	[5]	[6]	[7]
output	1		2	3	3	4	

Input	1	4	3	2	3	5	2

	[0]	[1]	[2]	[3]	[4]	[5]
Sum Count	0	0	2	3	5	7

	[1]	[2]	[3]	[4]	[5]	[6]	[7]
output	1		2	3	3	4	5

Input	1	4	3	2	3	5	2

	[0]	[1]	[2]	[3]	[4]	[5]
Sum Count	0	0	2	3	5	6

	[1]	[2]	[3]	[4]	[5]	[6]	[7]
output	1	2	2	3	3	4	5

Input	1	4	3	2	3	5	2

	[0]	[1]	[2]	[3]	[4]	[5]
Sum Count	0	0	1	3	5	6

	[1]	[2]	[3]	[4]	[5]	[6]	[7]
output	1	2	2	3	3	4	5

Array is now sorted

	[1]	[2]	[3]	[4]	[5]	[6]	[7]
output	1	2	2	3	3	4	5

Algorithm:

```
countingSort(array, n) // 'n' is the size of array
max = find maximum element in the given array
create count array with size maximum + 1
Initialize count array with all 0's
for i = 0 to n
find the count of every unique element and
store that count at ith position in the count array
for j = 1 to max
Now, find the cumulative sum and store it in count array
for i = n to 1
Restore the array elements
Decrease the count of every restored element by 1
end countingSort
```

Coding

```
// Store count of each character
    for (i = 0; arr[i]; ++i)
        ++count[arr[i]];

    // Change count[i] so that count[i] now contains
actual
    // position of this character in output array
    for (i = 1; i <= RANGE; ++i)
        count[i] += count[i - 1];

    // Build the output character array
    for (i = 0; arr[i]; ++i) {
        output[count[arr[i]] - 1] = arr[i];
        --count[arr[i]];
    }
// Copy the output array to arr, so that arr now
    // contains sorted characters
    for (i = 0; arr[i]; ++i)
        arr[i] = output[i];
}
```

Time Complexity: *O(n)*