Basic Skills 1

TAI, WEI HSUAN

August 6, 2025

Outline

- Sorting Algorithms
- Binary Search
- Fast Exponentiation
- Prefix Sum

Sorting Algorithms

Given a series of numbers, can you sort them in ascending order?
This is a very common problem in programming, and there are many algorithms to solve this problem.
There are many sorting algorithms, such as:

- Bubble Sort
- Selection Sort
- Insertion Sort
- Merge Sort
- Quick Sort
- Heap Sort
- Counting Sort
- Radix Sort
- Bucket Sort

Sorting Algorithms

In this class, we won't go into details about each sorting algorithm, instead, we will use the built-in sorting function in C++.

If you want to learn more about sorting algorithms, you can check out this link: `https://oi-wiki.org/basic/sort-intro/`

The built-in sorting function is based on the `std::sort` function

Code for Sorting Algorithms

```cpp
#include<algorithm>
int arr[10005], n;

sort(arr, arr+n); // Sort the array in ascending order

```

Using this function, we can sort an array in ascending order with the time complexity of $O(n \log n)$, where n is the number of elements in the array.

Another way to sort

If you want to sort an array in your own way, you can define a custom comparator function.
For example, if you want to sort an array in descending order, you can define a function like this:

```
bool cmp(int a, int b) {
    return a>b; // Return true if a is greater than b
}

sort(arr, arr+n, cmp); // Sort the array in descending order

// Or you can use a lambda function
sort(arr, arr+n, [](int a, int b) {
    return a > b;
});
```

Principle of Comparison function

The comparison function should return true if the first argument should come before the second in the final sorted order; otherwise, it should return false.

More about sorting

Give you a series of student scores, sort them by scores. If scores are the same, the sort by student ID in ascending order.

More about sorting

```
struct Student{
    int id; // Student ID
    int score; // Student score
};
bool cmp(Student a, Student b) {
    if(a.score==b.score)return a.id<b.id; // If scores are the same, sort by ID
    return a.score>b.score; // Sort by score in descending order
}
vector<Student> students;
sort(students.begin(), students.end(), cmp); // Sort the students by score and ID
```

Binary Search

Given a sorted array, can you find the position of a target value in the array?
Ofcourse, we can use a linear search to do that.

```
int linear_search(vector<int> arr, int target) {
    for(int i=0;i<arr.size();i++) {
        if(arr[i]==target) {
            return i; // Found the target at index i
        }
    }
    return -1; // Target not found
}
```

Time complexity?

Binary Search

We can use a concept of "cut in half"
For example, if you want to gurss a number between 1 and 100, you can start by guessing 50, if the answer is smaller than 50, you can cut the range to 1-49, if the answer is larger than 50, you can cut the range to 51-100.

Code for Binary Search

```
vector<int> arr;
int target;
int binary_search(int l, int r){
    if(l>r)return -1; // Not found
    int mid=l+r>>1;
    if(mid==target)return mid; // Found the target
    if(arr[mid]>target)return binary_search(l, mid-1); // Search in
    ↪ the left half
    return binary_search(mid+1, r); // Search in the right half
}
```

Time complexity?

Practice

- SPOJ - Binary Search

C++ have a built-in function for binary search

```
#include<algorithm>
vector<int> arr;
int target;
int idnex=lower_bound(arr.begin(), arr.end(),
↪  target)-arr.begin();

```

Summary

- We can use binary search to find the position of a target value in a sorted array with the time complexity of $O(\log n)$, where n is the number of elements in the array.
- C++ have a built-in function for binary search, which is `lower_bound` and `upper_bound`.

Fast Exponentiation

Given a number a and an exponent b, can you calculate a^b ?
Ofcourse we can use a linear search to do that.

```
int a, b, ans=1, mod=1e9+7;
for(int i=0;i<b;i++){
    ans=ans*a%mod; // Multiply a to ans, and take mod
}
cout<<ans; // Output the result
```

Time complexity?

Fast Exponentiation

Fast exponentiation is a technique to compute large powers of a number efficiently. The idea is to use the property:

$$a^b = \begin{cases} 1 & \text{if } b = 0 \\ a \cdot a^{b-1} & \text{if } b \text{ is odd} \\ (a^{b/2})^2 & \text{if } b \text{ is even} \end{cases}$$

Example

If you want to calculate 2^{15}, you can do it like this:

- $2^{15} = 2^{14} \cdot 2$
- $2^{14} = (2^7)^2$
- $2^7 = 2^6 \cdot 2$
- $2^6 = (2^3)^2$
- $2^3 = 2^2 \cdot 2$
- $2^2 = (2^1)^2$
- $2^1 = 2$

This way, we can reduce the number of multiplications needed to calculate 2^{15}.
To calculate a^b, we can reduce the time complexity to $O(\log b)$ by using the fast exponentiation technique.

Code for fast exponentiation

```
int fpow(int a, int b){
    int ans=1;
    while(b>0){
        if(b & 1)ans=ans*a%mod; // If b is odd, multiply a to ans
        a=a*a%mod; // Square a
        b>>=1; // Divide b by 2
    }
    return ans;
}
```

Practice

- CSES 1095 - Pure Exponentiation

Prefix Sum

Given an array with n elements adn q queries, can you calculate the sum of elements in a range $[l, r]$?
One simple way is to use a loop to calculate the sum.

```
vector<int> arr;
int q, l, r;
for(int i=0;i<q;i++){
    cin>>l>>r;
    int sum=0;
    for(int j=l;j<=r;j++){
        sum+=arr[j];
    }
    cout<<sum<<"\n";
}
```

Time complexity?

Prefix Sum

We can use a technique called prefix sum to solve this problem.
Set

$$p_i = \sum_{k=1}^{i} a_k$$

If we want to calculate the sum of elements in a range $[l, r]$, we can use the formula:

$$\sum_{k=l}^{r} a_k = \sum_{k=1}^{r} a_k - \sum_{k=1}^{l-1} a_k$$

Though we need $O(n)$ to preprocess the prefix sum array, we can reduce the time complexity to $O(1)$ for each query.

Code for Prefix Sum

```
vector<int> arr;
int pre[N], q, l, r;
for(int i=1;i<=n;i++){
    pre[i]=pre[i-1]+arr[i]; // Calculate the prefix sum
}
for(int i=0;i<q;i++){
    cin>>l>>r;
    cout<<pre[r]-pre[l-1]<<"\n"; // Output the sum of elements in the range [l, r]
}
```

Challenge

Given an array with n elements and q queries, each query has one number. Can you output a pair of indices (l, r) such that the sum of elements in the range $[l, r]$ is equal to the number in the query?

Practice

- CSES 1646 - Pure Prefix Sum