

# Cpp Templates

ZJUT 12

2021.3

# Contents

<b>1</b>	<b>Basic Algorithms</b>	<b>2</b>
1.1	Sortings and Applications . . . . .	2
1.1.1	Quick Sort . . . . .	2
1.1.2	Quick Find . . . . .	2
1.1.3	Merge Sort . . . . .	2
1.1.4	Inversion Pairs . . . . .	3
1.2	Binary Search . . . . .	3
1.2.1	Integer Dichotomy . . . . .	3
1.2.2	Float Dichotomy . . . . .	4
1.3	Ternary Search . . . . .	5
1.4	Non-negative Big Integer Calculations . . . . .	5
1.4.1	Big Integer Addition . . . . .	6
1.4.2	Big Integer Subtraction . . . . .	6
1.4.3	Big Integer Multiplies Integer . . . . .	7
1.4.4	Big Integer Divides Integer . . . . .	7
<b>2</b>	<b>Dynamic Programming</b>	<b>8</b>
<b>3</b>	<b>Data Structure</b>	<b>8</b>
<b>4</b>	<b>Graph Theory</b>	<b>8</b>
<b>5</b>	<b>Mathematics</b>	<b>8</b>
<b>6</b>	<b>Computational Geometry</b>	<b>8</b>
<b>7</b>	<b>Strings</b>	<b>8</b>
<b>8</b>	<b>Greedy</b>	<b>8</b>
<b>9</b>	<b>STL</b>	<b>8</b>

# 1 Basic Algorithms

## 1.1 Sortings and Applications

### 1.1.1 Quick Sort

Time Complexity:  $O(n \log n) \sim O(n^2)$

```
1 void qsort(int a[], int l, int r)
2 {
3     if (l == r) return;
4     int i = l - 1, j = r + 1, mid = a[l + r >> 1];
5     while (i < j)
6     {
7         do ++i; while (a[i] < mid);
8         do --j; while (a[j] > mid);
9         if (i < j) swap(a[i], a[j]);
10    }
11    qsort(a, l, j); qsort(a, j + 1, r);
12 }
```

### 1.1.2 Quick Find

Time Complexity:  $O(n)$

```
1 // Return the kth element in range [l,r].
2 int qfind(int a[], int l, int r, int k)
3 {
4     if (l == r) return a[l];
5     int i = l - 1, j = r + 1, mid = a[l + r >> 1];
6     while (i < j)
7     {
8         do ++i; while (a[i] < mid);
9         do --j; while (a[j] > mid);
10        if (i < j) swap(a[i], a[j]);
11    }
12    int len = j - l + 1;
13    if (len >= k) return qfind(a, l, j, k);
14    else return qfind(a, j + 1, r, k - len);
15 }
```

### 1.1.3 Merge Sort

Time Complexity:  $O(n \log n)$

```
1 int temp[];
2 void merge_sort(int a[], int l, int r)
```

```

3 {
4     if (l == r) return;
5     int mid = l + r >> 1;
6     merge_sort(a, l, mid); merge_sort(a, mid + 1, r);
7     int k = 0, i = l, j = mid + 1;
8     while (i <= mid && j <= r)
9     {
10         if (a[i] <= a[j]) temp[++k] = a[i++];
11         else temp[++k] = a[j++];
12     }
13     while (i <= mid) temp[++k] = a[i++];
14     while (j <= r) temp[++k] = a[j++];
15     for (int i = l, j = 1; i <= r; ++i, ++j) a[i] = temp[j];
16     return;
17 }

```

#### 1.1.4 Inversion Pairs

Time Complexity:  $O(n \log n)$

```

1 // Return the number of inversion pairs in range [l,r].
2 int temp[];
3 ll invPair(int a[], int l, int r)
4 {
5     if (l == r) return 0;
6     int mid = l + r >> 1;
7     ll ans = invPair(a, l, mid) + invPair(a, mid + 1, r);
8     int k = 0, i = l, j = mid + 1;
9     while (i <= mid && j <= r)
10    {
11        if (a[i] <= a[j]) temp[++k] = a[i++];
12        else temp[++k] = a[j++], ans += mid - i + 1;
13    }
14    while (i <= mid) temp[++k] = a[i++];
15    while (j <= r) temp[++k] = a[j++];
16    for (int i = l, j = 1; i <= r; ++i, ++j) a[i] = temp[j];
17    return ans;
18 }

```

## 1.2 Binary Search

Time Complexity:  $O(\log n)$

### 1.2.1 Integer Dichotomy

```

1 // Find one element in an ordered sequence.
2 function < int(int) > check = [&](int mid) {};

```

```

3  int l, r, ans = -1;
4  while (l <= r)
5  {
6      int mid = l + r >> 1, t = check(mid);
7      if (!t)
8      {
9          ans = mid; break;
10     }
11     else
12     {
13         if (t > 0) r = mid - 1;
14         else l = mid + 1;
15     }
16 }

1  // Find the maximum element satisfying some conditions.
2  function < bool(int) > check = [&](int mid) {};
3  int l, r, ans = l;
4  while (l <= r)
5  {
6      int mid = l + r >> 1;
7      if (check(mid)) ans = mid, l = mid + 1;
8      else r = mid - 1;
9  }

1  // Find the minimum element satisfying some conditions.
2  function < bool(int) > check = [&](int mid) {};
3  int l, r, ans = l;
4  while (l <= r)
5  {
6      int mid = l + r >> 1;
7      if (check(mid)) ans = mid, r = mid - 1;
8      else l = mid + 1;
9  }

```

### 1.2.2 Float Dichotomy

```

1  // Find the maximum value satisfying some conditions.
2  const double eps = 1e-8;
3  function < bool(double) > check = [&](double mid) {};
4  double l, r, ans = l;
5  while (r - l > eps)
6  {
7      double mid = l + (r - l) / 2;
8      if (check(mid)) ans = mid, l = mid;
9      else r = mid;
10 }

```

```

1 // Find the minimum value satisfying some conditions.
2 const double eps = 1e-8;
3 function < bool(double) > check = [&](double mid) {};
4 double l, r, ans = l;
5 while (r - l > eps)
6 {
7     double mid = l + (r - l) / 2;
8     if (check(mid)) ans = mid, r = mid;
9     else l = mid;
10 }

```

### 1.3 Ternary Search

Time Complexity:  $O(\log n)$

```

1 // Find the maximum in an unimodal function.
2 const double eps = 1e-8;
3 function < double(double) > f = [&](double x) {};
4 double l, r;
5 while (r - l > eps)
6 {
7     double m1 = l + (r - l) / 3.0;
8     double m2 = r - (r - l) / 3.0;
9     if (f(m1) > f(m2)) r = m2;
10    else l = m1;
11 }
12 // l is the answer.

1 // Find the minimum in an unimodal function.
2 const double eps = 1e-8;
3 function < double(double) > f = [&](double x) {};
4 double l, r;
5 while (r - l > eps)
6 {
7     double m1 = l + (r - l) / 3.0;
8     double m2 = r - (r - l) / 3.0;
9     if (f(m1) > f(m2)) l = m1;
10    else r = m2;
11 }
12 // r is the answer.

```

### 1.4 Non-negative Big Integer Calculations

Use strings to read in big integers and save each digit inversely in an vector.  
Answer is also saved inversely.

### 1.4.1 Big Integer Addition

Time Complexity:  $O(m + n)$

```
1 std::vector < int > add(std::vector < int > &A, std::vector < int
  ↪ > &B)
2 {
3     std::vector < int > temp; int t = 0;
4     for (int i = 0; i < A.size() || i < B.size(); ++i)
5     {
6         if (i < A.size()) t += A[i];
7         if (i < B.size()) t += B[i];
8         temp.push_back(t % 10);
9         t /= 10;
10    }
11    if (t) temp.push_back(t);
12    return temp;
13 }
```

### 1.4.2 Big Integer Subtraction

Time Complexity:  $O(m + n)$

```
1 // Use cmp(a, b) to detect whether string a is larger than string
  ↪ b.
2 // If so, swap a, b and print a '-'.
3 bool cmp(std::string &a, std::string &b)
4 {
5     if (a.length() != b.length()) return a.length() < b.length();
6     else
7         for (int i = 0; i < a.length(); ++i)
8             if (a[i] != b[i]) return a[i] < b[i];
9     return 0;
10 }
11 std::vector < int > sub(std::vector < int > &A, std::vector < int
  ↪ > &B)
12 {
13     std::vector < int > temp; int t = 0;
14     for (int i = 0; i < A.size(); ++i)
15     {
16         t = A[i] - t;
17         if (i < B.size()) t -= B[i];
18         temp.push_back((t + 10) % 10);
19         if (t < 0) t = 1; else t = 0;
20     }
21     // Remove leading zeros.
22     while (temp.size() > 1 && temp.back() == 0) temp.pop_back();
```

```

23     return temp;
24 }

```

### 1.4.3 Big Integer Multiplies Integer

Time Complexity:  $O(n)$

```

1  std::vector < int > mul(std::vector < int > &A, int b)
2  {
3      std::vector < int > temp; int t = 0;
4      for (int i = 0; i < A.size() || t; ++i)
5      {
6          if (i < A.size()) t += A[i] * b;
7          temp.push_back(t % 10);
8          t /= 10;
9      }
10     // Remove leading zeros.
11     while (temp.size() > 1 && temp.back() == 0) temp.pop_back();
12     return temp;
13 }

```

### 1.4.4 Big Integer Divides Integer

Time Complexity:  $O(n)$

```

1  // r means remainders.
2  std::vector < int > div(std::vector < int > &A, int b, int &r)
3  {
4      std::vector < int > temp;
5      for (int i = A.size() - 1; i >= 0; --i)
6      {
7          r = r * 10 + A[i];
8          temp.push_back(r / b);
9          r %= b;
10     }
11     reverse(temp.begin(), temp.end());
12     while (temp.size() > 1 && temp.back() == 0) temp.pop_back();
13     return temp;
14 }

```



**2   Dynamic Programming**

**3   Data Structure**

**4   Graph Theory**

**5   Mathematics**

**6   Computational Geometry**

**7   Strings**

**8   Greedy**

**9   STL**