



Fast IO in Java

```

1 public class Main {
2     public static void main(String[] args) throws IOException {
3         BufferedReader in = new BufferedReader(new InputStreamReader(System.
4             in));
5         PrintWriter out = new PrintWriter(new BufferedWriter(new
6             OutputStreamWriter(System.out)));
7         StringTokenizer st = new StringTokenizer(in.readLine());
8         int tests = Integer.parseInt(st.nextToken());
9         for (int test = 1; test <= tests; test++) {
10             st = new StringTokenizer(in.readLine());
11             int n = Integer.parseInt(st.nextToken());
12             ...
13         }
14         in.close();
15         out.close(); // don't forget me
16     }
17 }

```



GDB

```
1 $ gdb a.out
2 NO WARRANTY this program won't work !
3 >> run
4 >> run < input
5 # Once it runs
6 >> break 42
7 >> break gridland # Break the function gridland
8 >> break 42 if n == 2
9 # Once it is stopped
10 >> step # Goes inside functions
11 >> next # Doesn't
12 >> continue
13 >> bt
14 >> bt full
15 >> frame 3
16 >> info locals
17 >> quit
```


Array

- Keys : $\{0, \dots, n-1\}$
- Access in $\mathcal{O}(1)$
- Modify in $\mathcal{O}(1)$

With dynamic array (`ArrayList` or `std::vector`), n is not fixed.

- Add in amortized $\mathcal{O}(1)$
- Remove in $\mathcal{O}(n)$ if not at the end

Stack and Queue

Specific Linked List (a bit faster and limited capability)

Stack LIFO, Last in First out.

```
1 Stack<Integer> s = new Stack<Integer>();
2 q.push(1);
3 int top = q.peek(); // just watch
4 top = q.pop();
```

Queue FIFO, First in First out.

```
1 Queue<Integer> q = new LinkedList<Integer>();
2 q.add(1);
3 int first = q.peek(); // just watch
4 first = q.poll();
```


Classical Example II

- With DP, $\mathcal{O}(n^3)$
- If we only check the size when the rectangles are “closed”, $\mathcal{O}(n^2)$

```

1 class Rectangle {
2     int start_row;
3     int width;
4 }
5 Stack<Rectangle>[] dp = new Stack[ncol]; // + init them
6 int best = 0
7 for row = 0..nrow-1
8     int width = 0
9     for col = ncol-1..0 // to process width in $\bigoh(1)$
10         if free cell
11             width++;
12             int start_row = row
13             while dp[col].peek().width >= width
14                 Rectangle closed = dp[col].pop();
15                 best = Math.max(best, closed.size(row))
16                 start_row = Math.min(row, closed.start_row)
17             dp[col].add(new Rectangle(start_row, width))
18         else
19             while !dp[col].empty()
20                 best = Math.max(best, dp[col].pop().size(row))
21                 width = 0
22 for col = ncol-1..0 // don't forget !!
23     close all rectangles of dp[col] -> update best // Lines 19,20

```


Trap problem – 10226

- If you want to do it in C, it will take you 1 h, you loose a precious time.
- With the STL, you kill it in 5 min !

Input

Don't Panic
Mostly Harmless
42
Don't Panic
The Hitchhiker's Guide

Output

```
42 20.0000
Don't Panic 40.0000
Mostly Harmless 20.0000
The Hitchhiker's Guide 20.0000
```

- `System.out.printf("%s %.4f\n", s, p)`
- Exactly 4 decimals `cout << setiosflags(ios::fixed)<< setprecision(4)`
- Strings with spaces `std::getline`

Unique key

Different elements can have the same key

15/53

Hash Map II

Collision

But there can be collision so it is $\mathcal{O}(1)$ on average! In case of collision, 2 solutions

- Store in each bucket all the collisions;
- Probe a empty spot (linear probing, quadratic probing, ...).

It is only $\mathcal{O}(1)$ on average if the load factor is good (e.g. $\ll 1$). When load factor get close to 1, increase N .

Binary Search Tree II

To get $\mathcal{O}(\log(n))$

- Balanced
- Comparison in $\mathcal{O}(1)$ (strings have $\mathcal{O}(m \log(n))$ where m is their length)
- Balancing operation in $\log(n)$

Balanced trees are tricky to code! Use `std::set`, `TreeSet` and `std::map`, `TreeMap`!

When to use a BST or a hash map

- When you want an array to be indexed by another thing than an int
- When an array is too small and you don't need all
- (*BST only*) When you want a structure to be
 - ▶ an array to get $\mathcal{O}(\log(n))$ binary search access time
 - ▶ a linked list to get $\mathcal{O}(1)$ insertion time

In the STL

```

1 TreeMap<String, Integer> map = new TreeMap<String, Integer>();
2 map.put("Test", 42);
3 map.get("Test"); // -> 42;
4 map.containsKey("Test"); // -> true
5 map.containsKey("Toto"); // -> false
6 for (Map.Entry<String, Integer> entry : map.entrySet()) {
7     String test = entry.getKey();
8     int fortytwo = entry.getValue();
9 }
10
11 std::map<std::string, int> map();
12 map["Test"] = 42;
13 map["Test"]; // -> 42
14 map.count("Test"); // -> 1
15 map.count("Toto"); // -> 0 (only 0 or 1 since not multimap)
16 for (std::map<std::string, int>::iterator it=map.begin(); it!=map.end
    (); ++it)
17     std::cout << it->first << " => " << it->second << '\n';

```

Interesting example

Longest Increasing Subsequence : LIS, a classic !

- Find an $\mathcal{O}(n^2)$ algorithm.
- Improve it to $\mathcal{O}(n \log(n))$.
- Solve problem 481 in uva with it (even if $\mathcal{O}(n^2)$ is enough).

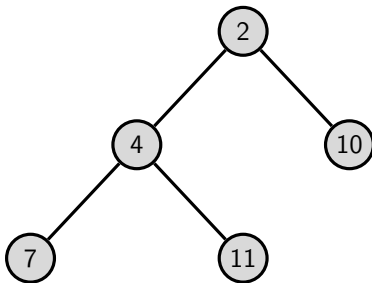
Let's do a problem, shall we?

Do problem 10954 from uva !

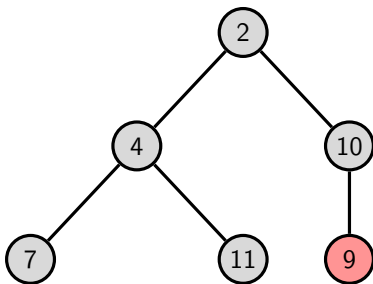
Heap

If we are only interested by the minimum, the BST is simplified and much faster and easier to code. Still $\mathcal{O}(\log(n))$ though.

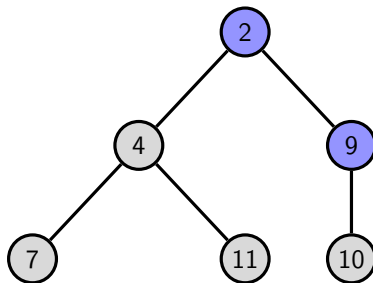
The tree can be always complete!



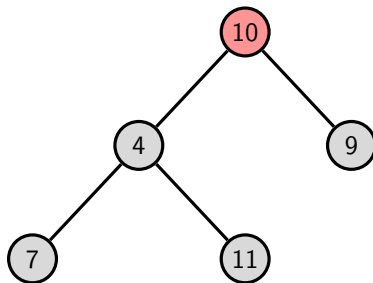
○○○○



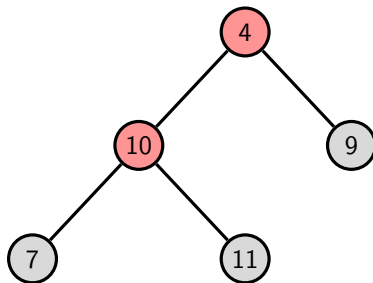
Push IV



○○○○



Pop IV




```

graph TD
    4((4)) --> 7((7))
    4 --> 9((9))
    7 --> 10((10))
    7 --> 11((11))
  
```

Implémentation I

```
1  int hsize;
2  int hid[MAX]; // give id from index
3  int hval[MAX]; // give val from index
4  int hlookup[MAX]; // give index from id
5
6  void heap_swap (int a, int b) { // a and b are the indexes
7      int tmp = hval[a];
8      hval[a] = hval[b];
9      hval[b] = tmp;
10
11     hlookup[hid[a]] = b;
12     hlookup[hid[b]] = a;
13
14     tmp = hid[a];
15     hid[a] = hid[b];
16     hid[b] = tmp;
17 }
18
19 void heap_up (int a) { // a is the index
20     int up = (a-1)/2;
21     if (0 < a && hval[a] < hval[up]) {
22         heap_swap(a, up);
23         heap_up(up);
24     }
25 }
26
```

Implémentation II

```
27 void heap_down (int a) { // a is the index
28   int left = 2*a+1, right = 2*a+2;
29   if (left < hsize && (hsize <= right || hval[left] < hval[right]) &&
        hval[left] < hval[a]) {
30     heap_swap(a, left);
31     heap_down(left);
32   }
33   else if (right < hsize && hval[right] < hval[a]) {
34     heap_swap(a, right);
35     heap_down(right);
36   }
37 }
```

A wild problem appear

n objects $0, \dots, n-1$ are initially alone in a set. We have two types of query

- We merge 2 sets designated by 2 respective members
- We ask if 2 objects are in the same set

	$\{1\}$	$\{2\}$	$\{3\}$	$\{4\}$	$\{5\}$
$(1, 5)$	$\{1, 5\}$	$\{2\}$	$\{3\}$	$\{4\}$	
$(2, 4)$	$\{1, 5\}$	$\{2, 4\}$	$\{3\}$		
$(2, 3)$	$\{1, 5\}$	$\{2, 3, 4\}$			
$(3, 4)$	$\{1, 5\}$	$\{2, 3, 4\}$			
$(4, 5)$	$\{1, 2, 3, 4, 5\}$				

Union find solution I

```
1 class UnionFind {
2     int rank[MAX_N];
3     int leader[MAX_N];
4     UnionFind(int n) {
5         memset(rank, 0, n * sizeof(int));
6         for(int i = 0; i < n; i++) leader[i] = i;
7     }
8     int find(int a) {
9         if(a != leader[a])
10             leader[a] = find(leader[a]);
11         return leader[a];
12     }
13     void union(int a, int b) {
14         int leaderA = find(a);
15         int leaderB = find(b);
16         if(leaderA == leaderB) return;
17         if(rank[leaderA] > rank[leaderB]) {
18             union(leaderB, leaderA); return;
19         }
20         leader[leaderA] = leaderB;
21         if (rank[leaderA] == rank[leaderB])
22             rank[leaderB]++;
23     }
24 };
```


Union find solution II

Complexity

time Amortized $\mathcal{O}(\alpha(n))$.

space $\mathcal{O}(n)$.

Example : gridland II

5 5 15

a 1 1

a 2 3

a 2 4

a 2 5

a 3 3

a 4 2

a 5 2

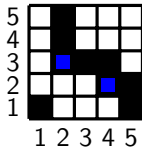
a 5 1

c 2 5 5 1

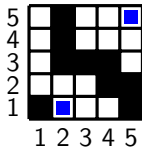
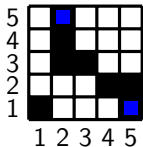
$$>> 0$$

a 4 3

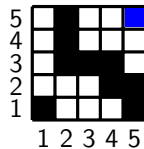
c 2 3 4 2

$$>> 1$$


c 2 1 5 5

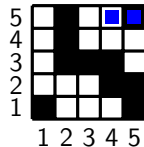
$$>> 0$$


c 4 4 4 4

$$>> 1$$


a 5 5

c 4 5 5 5

$$>> 0$$


Segment Tree I

- Dynamic Range Minimum Query
- Dynamic Range Sum Query (Fenwick Tree is simpler)
- Dynamic Range Anyfunction Query

```

1  class SegmentTree {
2      int[] st, A;
3      int n;
4      int left (int p) { return p << 1; }
5      int right(int p) { return (p << 1) + 1; }
6
7      void build(int p, int L, int R) {
8          if (L == R)
9              st[p] = L; // or R
10         else {
11             int mid = (L + R) / 2;
12             build(left(p), L, mid);
13             build(right(p), mid + 1, R);
14             int p1 = st[left(p)], p2 = st[right(p)];
15             st[p] = (A[p1] <= A[p2]) ? p1 : p2;
16         } }
17
18     int rmq(int p, int L, int R, int i, int j) { // O(log n)
19         if (i > R || j < L) return -1; // outside query range
20         if (i <= L && R <= j) return st[p]; // inside query range

```

Segment Tree II

```

21     int mid = (L + R) / 2;
22     int p1 = rmq(left(p) , L      , mid, i, j);
23     int p2 = rmq(right(p), mid + 1, R  , i, j);
24
25     if (p1 == -1) return p2;           // outside query  range
26     if (p2 == -1) return p1;
27     return (A[p1] <= A[p2]) ? p1 : p2; }
28
29 int update(int p, int L, int R, int i, int j, int v) {
30     if (i > R || j < L)                 // outside update range
31         return st[p];
32     //if (i <= L && R <= j) // could be lazy here !! Depends on
        application
33     if (L == R) {
34         A[i] = v;
35         return st[p] = L; // or R
36     }
37     int mid = (L + R) / 2;
38     int p1 = update(left(p) , L      , mid, i, j, v);
39     int p2 = update(right(p), mid + 1, R  , i, j, v);
40     return st[p] = (A[p1] <= A[p2]) ? p1 : p2;
41 }
42
43 public:
44
45 SegmentTree(int[] _A) {
46     A = _A; n = A.length;

```

Segment Tree III

```
47     st = new int[4 * n];
48     for (int i = 0; i < 4 * n; i++) st[i] = 0;
49     build(1, 0, n - 1);
50 }
51 int rmq(int i, int j) { return rmq(1, 0, n - 1, i, j); }
52 int update_point(int i, int v) {
53     return update(1, 0, n - 1, i, i, v); }
54 int update_interval(int i, int j, int v) {
55     return update(1, 0, n-1, i, j, v); }
56 };
```

- IOI-2013 day 2 : “game”
- <http://codeforces.com/problemset/problem/474/E>

Fenwick Tree I

Dynamic Range Sum Query

What are the numbers smaller that 1011001000?

- 1011000???
- 1010???????
- 100???????
- 0?????????

$$\text{rsq}(1011001000) = \text{ft}(1011001000) + \text{ft}(1011000000) \\ + \text{ft}(1010000000) + \text{ft}(1000000000)$$

```

1  adjust(01011001000,1):
2      ft[01011010000]++
3      ft[01011100000]++
4      ft[01100000000]++
5      ft[10000000000]++

```

```

1  x          0000000000000000000010010110100000
2  ~x         1111111111111111111101101001011111
3  -x or (~x)+1 1111111111111111111101101001100000
4  x & (-x)    0000000000000000000000000000000100000

```


Fenwick Tree II

```

1  class FenwickTree {
2      int *ft;
3      int n;
4      int LSOne(int S) { return (S & (-S)); }
5      public:
6      FenwickTree(int n) { // ignore index 0
7          this->n = n;
8          ft = new int[n+1];
9          for (int i = 0; i <= n; i++) ft[i] = 0;
10     }
11     int rsq(int b) {          // returns RSQ(1, b)      PRE 1 <= b <= n
12         int sum = 0; for (; b > 0; b -= LSOne(b)) sum += ft[b];
13         return sum;
14     }
15     int rsq(int a, int b) { // returns RSQ(a, b)      PRE 1 <= a,b <= n
16         return rsq(b) - (a == 1 ? 0 : rsq(a - 1));
17     }
18     void adjust(int k, int v) { // n = ft.size() - 1 PRE 1 <= k <= n
19         for (; k <= n; k += LSOne(k)) ft[k] += v;
20     }
21 };

```

Fenwick Tree III

Exemples :

- NWERC 2011 Problem C
- <http://codeforces.com/contest/504/problem/B>
- <http://codeforces.com/problemset/problem/459/D>

Sparse Table

Static Range Minimum Query

$m[i][j]$ stores the smallest of $[i; i + 2^j[$.

$\min([a; b[) = \min(m[a][k], m[b - 2^k][k])$ for k such that

$$2^{k-1} < b - a \leq 2^k$$

- 53/53