Basic Data Structures
C++ STL

Computational Complexity

- Complexity = Resources required
- Time
    - TLE
- Space
    - MLE
- Description
    - Too late
- Communication

Asymptotic Notation

- We often don't need precise bounds on complexity.
- For large enough n:
  - T(n)=O(f(n)): $T(n) \le c_1 f(n)$ = means ∈
  - T(n)=Ω(f(n)): $T(n) \ge c_2 f(n)$
  - T(n)=Θ(f(n)): $c_2 f(n) \le T(n) \le c_1 f(n)$
  - T(n)=o(f(n)): $\lim_{n\to\infty} \frac{T(n)}{f(n)} = 0$
  - T(n)=ω(f(n)): $\lim_{n\to\infty} \frac{f(n)}{T(n)} = 0$

small-ω	big-Ω	Θ	big-O	Small-o
>	≥	=	≤	<

Growth of Function

- Factorial: $O(n!)$
- Exponential: $O(c^n)$
- Polynomial: $O(n^k)$
  - Cubic: $O(n^3)$
  - Square: $O(n^2)$
- Linearithmic: $O(n\log n)$
- Linear: $O(n)$
- Logarithmic: $O(\log n)$
- Constant: $O(1)$

STL

- Standard template library
- Very useful in competitive programming
- Composition
  - Containers
  - Iterators
  - Algorithms
  - Functors
- Will cover a small part of STL in this lecture

Useful Containers in Contest

- pair
- vector
- list
- queue
- deque
- priority_queue
- set
- map

pair

- Couple two values into a pair
- pair<string,int> p("abc",123);
- make_pair(string("abc"),123);
- p.first // access the first element
- p.second // access the second element
- Compare the field “first” first, then compare the field “second.”

vector

- An array cannot change its size but a vector can.
- vector is still fast enough in general.
- vector<int> a;   //create a vector initially empty
- vector<int> a(n);   //create a vector of n integers
- vector<int> a(n,5); // create a vector of n integers
  // and the integers are all 5
- k-th element of a: a[k]

vector

- a.back() // the last element of a
- a.push_back(x); // append x to vector a
- a.pop_back(); // remove the last element of a

vector as a stack

- LIFO: last-in first-out
- vector<int> s; // make an empty stack s of integers
- s.empty() // test if s is empty
- s.size() // number of elements in s
- s.push_back(x) // insert x into s
- s.pop_back() // remove the last inserted element
- s.back() // the top element in the stack
- class stack in STL is slower than vector.

queue

- FIFO: first-in first-out
- queue<int> q;   // make an empty queue of integers
- q.empty() // test if q is empty
- q.size() // number of elements in q
- q.push(x) // insert x into q
- q.pop() // remove an element from q
- q.front() // the next element to be popped
- q.back() // the last element pushed

priority_queue

- Greatest first out
- priority_queue<int> pq; // make an empty priority
  // queue of integers
- pq.empty()
- pq.size()
- pq.push()
- pq.pop()
- pq.top() // get the first element

set (Ordered Set)

- Stores unique elements in certain order
- set<int> s; //make an empty set of integers
    - s.empty()
    - s.size()
- s.insert(x) // insert x into s
- s.erase(x) // remove x from s
- s.count(x) // number of copies of x in s
- *s.begin() // first element in s
- *s.rbegin() // last element in s

map (Ordered Map)

- map a key into a unique value.
- Array is a kind of map. 0, …, n-1 are keys of int[n].
- map supports non-consecutive keys
- map<string,int> cnt; // initialize an empty map
  // from string to int
- cnt.empty()
- cnt.size()
- cnt.count(x) // number of elements having key x

map (Ordered Map)

- cnt["key"] = x // map "key" into x
- y = cnt["key"] // get the value from "key"
- *cnt->begin() // the smallest key-value pair
  - cnt->begin()->first // key of the smallest element
  - cnt->begin()->second // value of the smallest element
- *cnt->rbegin() // the greatest key-value pair
  - cnt->rbegin()->first
  - cnt->rbegin()->second

Useful Algorithms in Contest

- sort
- lower_bound
- upper_bound
- next_permutation

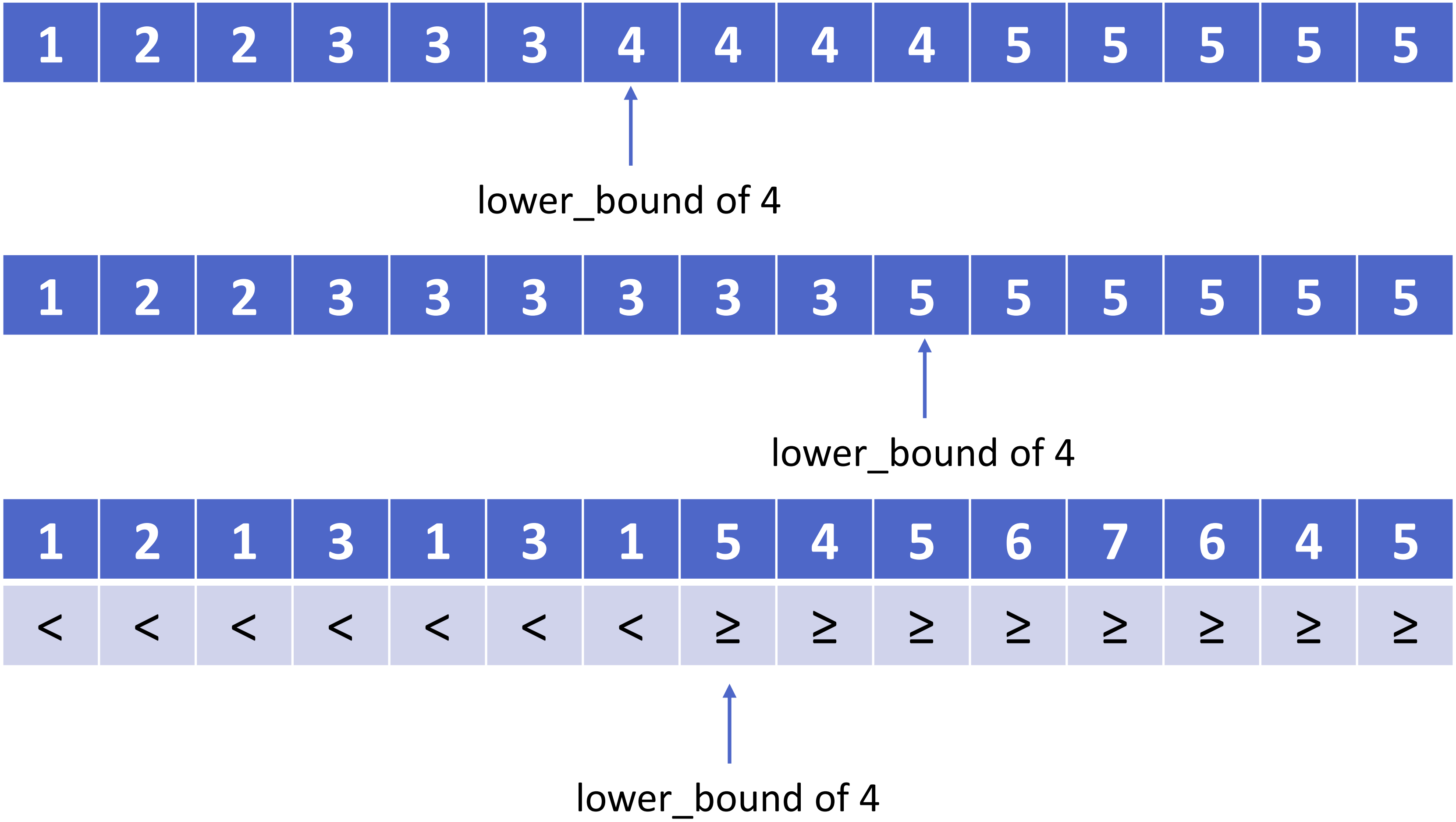
sort()

- Sorting elements into ascending order
- sort(a,a+100); // sort int a[100]
- sort(v.begin(), v.end()); // sort vector v
- The elements must be comparable
  - int and string are comparable
  - public bool operator<(const T& rhs) const
- In most cases, sort() is fast enough.
- Time complexity: O(nlogn)

lower_bound

- lower_bound(v.begin(), v.end(), x)
  - Finding the first position that is not <x in vector v
  - Returns an iterator
  - Works only if all elements are not <x after that position. For example, v is sorted in ascending order.
- Implementation: variant binary search
- Time complexity: O(logn)

lower_bound



1	2	2	3	3	3	4	4	4	4	5	5	5	5	5

lower_bound of 4

1	2	2	3	3	3	3	3	3	5	5	5	5	5	5

lower_bound of 4

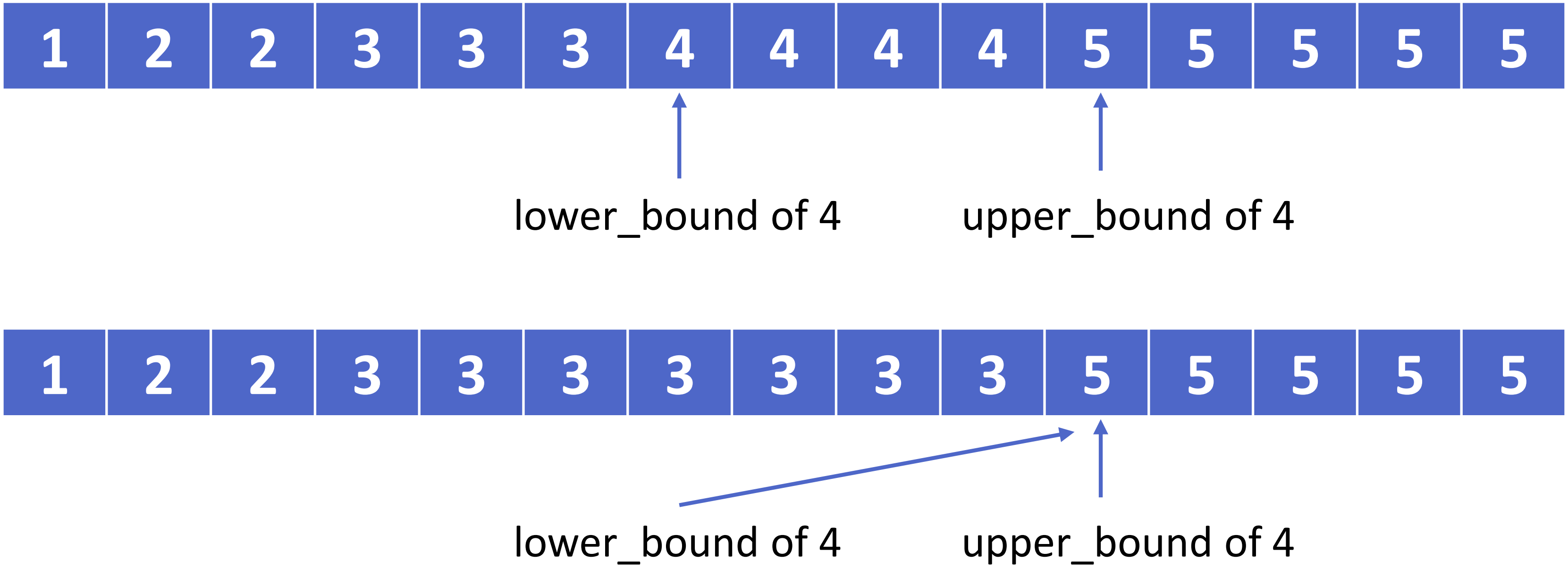
1	2	1	3	1	3	1	5	4	5	6	7	6	4	5
<	<	<	<	<	<	<	≥	≥	≥	≥	≥	≥	≥	≥

lower_bound of 4

upper_bound

- upper_bound(v.begin(), v.end(), x)
  - Finding the first position that is not ≤x in vector v
  - Returns an iterator
  - Works only if all elements are not ≤x after that position. For example, v is sorted in ascending order.
- Implementation: variant binary search
- Time complexity: O(logn)

lower_bound and upper_bound



1	2	2	3	3	3	4	4	4	4	5	5	5	5	5

lower_bound of 4

upper_bound of 4

1	2	2	3	3	3	3	3	3	3	5	5	5	5	5

lower_bound of 4

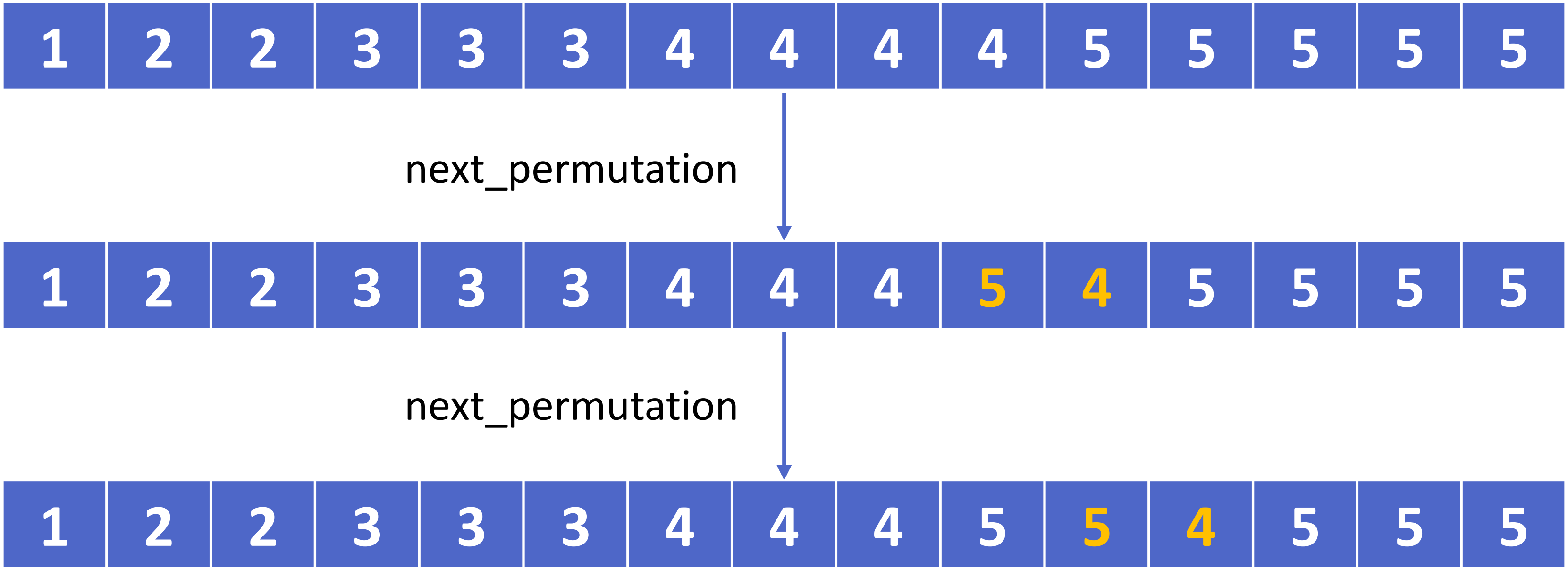
upper_bound of 4

next_permutation

Might be too slow in some application!

- Permutation: reordering of a sequence
- Lexicographic order of two permutation p and q
  - p < q if there exists i such that p[i] < q[i] and p[j] = q[j] for every j < i.
- Next permutation of p: the minimum of all permutation greater than p.
- next_permutation:
  - If the input has next permutation, then modify the input into its next permutation and return true.
  - Otherwise, return false.

Next Permutation



1 2 2 3 3 3 4 4 4 4 5 5 5 5 5

next_permutation

1 2 2 3 3 3 4 4 4 5 4 5 5 5 5

next_permutation

1 2 2 3 3 3 4 4 4 5 5 4 5 5 5