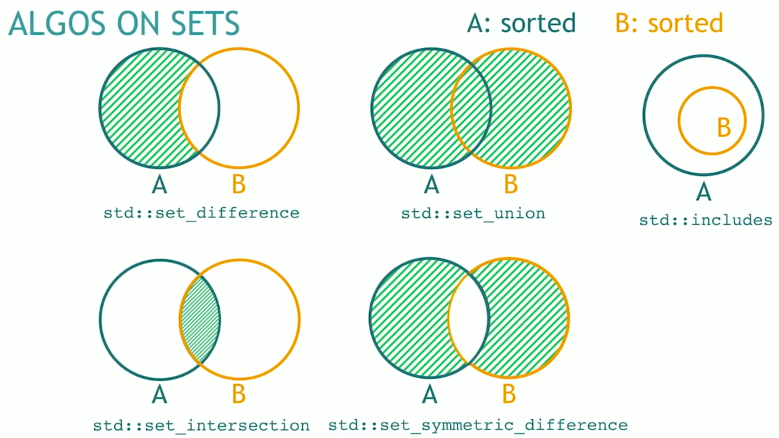
* Global array can have greater size (around 107) than local
* 1LL << k
* int mid = l + (r – l) / 2
* use emplace\_back instead of push\_back
* ios\_sync\_with\_stdio(0)
* {a, b} instead of make\_pair(a, b)
* Declare complicated iterators using auto
* for (auto x: container)
* #include <bits/stdc++.h>

STL algorithms

* Heaps
  + make\_heap(v.begin(), v.end(), cmp)
    - makes a heap from a range (children of node i are i \* 2 and i \* 2 + 1)
  + push\_heap(v.begin(), v.end(), cmp)
    - given a heap in range [begin, end – 1], it extends the range considered a heap to [begin, end]
  + pop\_heap(v.begin(), v.end(), cmp)
    - moves the first element (maximum / minimum) to the last, reduce range from [begin, end] to [begin, end – 1], while still being a heap
  + sort\_heap(v.begin(), v.end(), cmp)
    - sort the elements in heap range [begin, end] in ascending order, no longer a heap
    - repeatedly calls pop\_heap()
* Sorting
  + sort(v.begin(), v.end(), cmp)
    - sort the range [begin, end]
  + partial\_sort(v.begin(), v.begin() + m, v.end(), cmp)
    - pick the m smallest elements and sort them
    - Example: 5 4 3 2 1 🡪 partial\_sort (m = 3) 🡪 1 2 3 5 4
  + nth\_element(v.begin(), v.begin() + m, v.end(), cmp)
    - rearranges elements such that the nth element is the elements that it would be if the array was sorted
    - elements on the left are smaller, elements on the right are larger
  + inplace\_merge(v.begin(), v.begin() + m, v.end(), cmp)
    - merges two sorted ranges [begin, m – 1] and [m, end] into a sorted range [begin, end]
* Partitioning
  + partition(v.begin(), v.end(), bool func)
    - rearranges elements in [begin, end] such that all elements for which bool func return true precedes others that return false
  + partition\_point(v.begin(), v.end(), bool func)
    - returns an iterator to the first element that returns false for bool func
* Permutations
  + rotate(v.begin(), v.begin() + m, v.end())
    - rotate the elements in range [begin, end] such that the mth element becomes the new first element
  + shuffle(v.begin(), v.end(), default\_random\_engine(seed))
    - randomly shuffle elements in [begin, end]
  + next\_permutation(v.begin(), v.end(), cmp)
    - rearranges elements in [begin, end] into the next lexicographically greater permutation
    - returns false if reach the end and the original permutation is the greatest
  + prev\_permutatiom(v.begin(), v.end(), cmp)
    - similar to above but returns the previous permutation
  + reverse(v.begin(), v.end())
    - reverse the elements in range [begin, end]
* Runes
  + stable\_\*(v.begin(), v.end(), cmp)
    - stable\_sort, stable\_partition
    - keeps the relative order of elements
    - time complexity same as non-stable counterparts
  + is\_\*(v.begin(), v.end(), cmp / bool func)
    - is\_sorted, is\_partitioned, is\_heap
  + is\_\*\_until(v.begin(), v.end(), cmp / bool func)
    - is\_sorted\_until, is\_partitioned\_until, is\_heap\_until
    - checks if it is \* until a point
* Numeric algorithms
  + count(v.begin(), v.end(), x)
    - returns how many elements are equal to x in the range
  + count\_if(v.begin(), v.end(), bool func)
    - returns how many elements returns true in the range
  + accumulate(v.begin(), v.end(), init, op)
    - returns the total sum / product / result after repeatedly applying op
  + partial\_sum(v.begin(), v.end(), v.begin(), op)
    - writes the partial sums of elements in [begin, end] by applying op to v2.begin()
* Binary search
  + binary\_search(v.begin(), v.end(), x, cmp)
    - returns true if x is in the range
  + lower\_bound(v.begin(), v.end(), x, cmp)
    - returns an iterator pointing to the first element in the range that is >= x
  + upper\_bound
    - returns an iterator pointing to the first element in the range that is > x
* Minimum and maximum
  + min(a, b, cmp) max(a, b, cmp)
  + min\_element(v.begin(), v.end(), cmp) max\_element(v.begin(), v.end(), cmp)
* Algorithms on sets (sorted ranges)
  + set\_difference(a.begin(), a.end(), b.begin(), b.end(), v.begin())
    - writes elements of a that doesn’t belong in b to v.begin()
  + set\_intersection, set\_union, set\_symmetric\_difference, includes (returns bool)
* Value modifiers
  + fill(v.begin(), v.end(), x)
  + generate(v.begin(), v.end(), func)
  + iota(v.begin(), v.end(), x)
  + replace(v.begin(), v.end(), x, y)
* Structure modifiers
  + v.erase(remove(v.begin(), v.end(), x), v.end())
  + v.erase(unique(v.begin(), v.end()), v.end())