# Binary Search

[**PROBLEM**] Given a sorted array Where all elements appear 2 times except 1 element. Find the element that occurs once.

*vector*<int> A = { 1, 1, 3, 3, 5, 5, 6, 6, 7, 7, 8, 8, 9, 9, 10, 10, 1000, 1000, 100000, 100000, 99999999 };

|  |
| --- |
| **APPROACH 1:**  TC: NlogN + N/2 logN/2 = O(NlogN) [Map is BBST]  SC: N/2 = O(N)   1. Build a Map 2. Find a Key Whose frequency is 1 |
| **APPROACH 2:**  TC: N [Set is BBST or HashTabIe]  SC: 2 = O(1)   1. Use a Set 2. For each element    1. If there in the set       1. Remove it       2. if only One element is Left          1. That is the answer    2. else       1. Insert it 3. Only element in set is the answer |
| **APPROACH 2.1:**  TC: N [Set is BBST or HashTabIe]  SC: 1 = 0(1)   1. Use a Set 2. For each element    1. If there in the set       1. Remove it    2. else       1. If set already has an element          1. That ele in set is the answer       2. Insert it 3. Only element in set is the answer |
| **APPROACH 3:**  Compare Adj elements (only even index with next index, be careful of bounds check)  Use Two Pointers  Freq Approach  TC: N  SC: 1   1. curEIem=a[0], curFre=1 2. for i=[1, n)    1. if a[i] is same as curEIe       1. curFre++    2. else       1. if(curFre==1) return curEle       2. curE = a[i]       3. curF = 1 |
| **APPROACH 4:**  TC: N  SC: (var count = O) = O(1)  XOR all elements of array. |
| **APPROACH 5:**  TC: logN  SC:  **Binary Search** |

## Binary Search Code

[**PROBLEM**] Given a sorted array in ***descending order***, and a key K, all elements are unique, return index of K in the array. -1 if K is not found in the array.

### Implement Binary Search Iterative

TC: O(IogN)

SC: O(1)

|  |
| --- |
| int findK(*vector*<int> a, int k) {  int l = 0, h = a.*size*() - 1;  int m;  while (l <= h) {  m = (l + (h - l) / 2);  if (a[m] == k) return m;  if (a[m] > k) l = m + 1;  else h = m - 1;  }  return -1;  } |

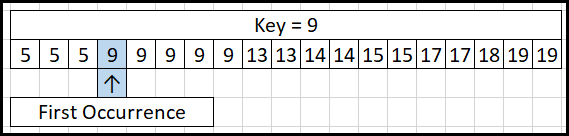
### Implement Binary Search Recursive

TC: logN [ T(N) = T(N/2) + 0]

SC: IogN [Recursive call stack]

|  |
| --- |
| int findK(*vector*<int>& a, int l, int r, int k) {  if (l > r) return -1;  int m = (l + (r - l) / 2);  if (a[m] == k)  return m;  if (a[m] < k)  return findK(a, l, m-l, k);  return findK(a, m + l, r, k);  }  int findK(*vector*<int>& a, int k) {  return findK(a, 0, a.*size*() - 1, k);  }  int main(void) {  *vector*<int> arr = { 99, 88, 18, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1};  *cout* << findK(arr, 88) << "\n";  return 0;  } |

[**PROBLEM**] Given sorted (asc) array having duplicates, and a key K. find first occurrence index of K, or -1 if K is not found.



*vector*<int> arr = { 5, 5, 5, 9, 9, 9, 9, 9, 13, 13, 14, 14, 15, 15, 17, 17, 18, 19, 19 };

|  |
| --- |
| int firstIndex(*vector*<int> a, int k) {  int low = 0, high = a.*size*() - 1;  int res = -1;  while (low <= high) {  int mid = low + (high - low) / 2;  if (k < a[mid])  high = mid - 1;  else if (k > a[mid])  low = mid + 1;  else {  res = mid;  high = mid - 1;  }  }  return res;  }  int main(void) {  *vector*<int> arr = { 5, 5, 5, 9, 9, 9, 9, 9, 13, 13, 14, 14, 15,  15, 17, 17, 18, 19, 19 };  *cout* << firstIndex(arr, 9) << "\n";  return 0;  } |

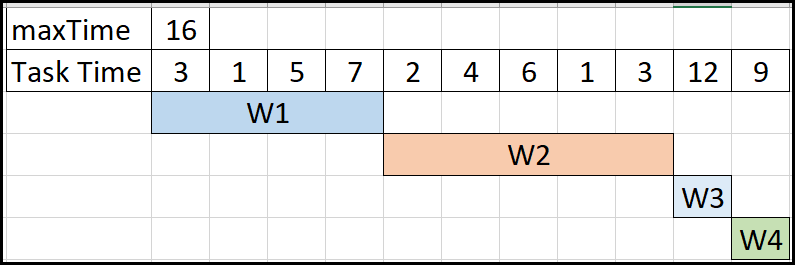
[**PROBLEM**] Given sorted (asc) array having duplicates, and a key K. find last occurrence index of K, or -1 if K is not found.

|  |
| --- |
| int lastIndex(*vector*<int> a, int k) {  int low = 0, high = a.*size*() - 1;  int res = -1;  while (low <= high) {  int mid = low + (high - low) / 2;  if (k < a[mid])  high = mid - 1;  else if (k > a[mid])  low = mid + 1;  else {  res = mid;  low = mid + 1;  }  }  return res;  } |

[**PROBLEM**] Count number of occurrences of K in sorted array a in logN time.

|  |
| --- |
| int countOccurences(*vector*<int>& a, int k) {  int li = lastIndex(a, k);  if (li == -1) return 0;  int fi = firstIndex(a, k);  return li - fi + 1;  } |

[**PROBLEM**] Can complete tasks With n workers? maXTimePerWorker is fixed. And we are given a count of workers. A worker can pick only consecutive tasks. Workers can remain idle also (not that all workers have to be occupied)



|  |
| --- |
| bool canComplete(*vector*<int> taskTimes, int workers, int maXTimePerWorker) {  int c = 1;  int sum = 0;  for (int i = 0; i < taskTimes.*size*(); i++) {  if (taskTimes[i] > maXTimePerWorker) return false;  if (sum + taskTimes[i] > maXTimePerWorker) {  c++;  sum = taskTimes[i];  }  else {  sum = sum + taskTimes[i];  }  if (c > workers) return false;  }  return (c <= workers);  } |

[**PROBLEM**] GIVEN N tasks and W workers, and each worker can pick only consecutive tasks. Find the min time in which you can complete the tasks. -1 if it is not possible.