Assignment 1

**Question 1:** Write a program to implement Quick sort

**Algorithm:**

Algorithm for method main():

Start

Step 1: Enter an array and store it in an integer array arr.

Step 2: Call subroutine sort with 0 and arr.size() - 1 as parameters.

Step 3: Print the sorted array.

Stop

Algorithm for subroutine sort():

Start

Step 1: Accept an integer array arr, and two integer variables low and high as parameters.

Step 2: if low < high

1. Call subroutine partition with arr, low, high as parameters and store the result in x.
2. Call subroutine sort with arr, low, x-1 as parameters
3. Call subroutine sort with arr, x+1, high as parameters.

Stop

Algorithm for subroutine partition():

Start

Step 1: Accept an integer array arr and integer variables low and high as parameters.

Step 2: pivot <- arr[high], i <-- low – 1

Step 3: for j = low to high – 1

1. if arr[j] < pivot
   1. i <-- i + 1
   2. swap arr[i], arr[j]

Step 4: swap arr[i+1], arr[high]

Step 5: return i + 1

Stop

**Code:**

#include <iostream>

#include <vector>

#include <iterator>

#include <algorithm>

template<typename T, typename Comp>

int partition(std::vector<T> &arr, int low, int high, Comp cmp)

{

auto [pivot, i] = std::make\_pair(arr[high], low-1);

for(int j = low; j < high; j++)

if(cmp(arr[j], pivot))

{

i++;

std::swap(arr[i], arr[j]);

}

std::swap(arr[i+1], arr[high]);

return i+1;

}

template<typename T, typename Comp>

void sort(std::vector<T> &arr, int low, int high, Comp cmp)

{

if(low < high)

{

int x = partition(arr, low, high, cmp);

sort(arr, low, x-1, cmp);

sort(arr, x+1, high, cmp);

}

}

int main()

{

std::vector<int> arr = {std::istream\_iterator<int>(std::cin), std::istream\_iterator<int>()};

sort<int>(arr, 0, arr.size() - 1, [](int x, int y){return x<y;});

std::cout<<"Sorted array:\n";

std::for\_each(arr.begin(), arr.end(), [](const int &x){std::cout<< x << '\n';});

}

**Input:** 1 6 2 5 3 4

**Output:** Sorted array:

1

2

3

4

5

6

**Question 2:** Write a program to implement merge sort algorithm

**Algorithm:**

Algorithm for method main():

Start

Step 1: Enter the numbers and store it in an integer array arr.

Step 2: Call subroutine sort() with arr, 0, arr.size() - 1 as parameters

Step 3: Print the sorted array

Stop

Algorithm for subroutine sort():

Start

Step 1: Accept an integer array arr and integer variables low and high as parameters

Step 2: if low < high

1. mid <-- ( low + high ) / 2
2. Call subroutine sort() with arr, low, mid as parameters
3. Call subroutine sort() with arr, mid + 1, high as parameters
4. Call subroutine merge() with arr, low, high as parameters

Stop

Algorithm for subroutine merge():

Start

Step 1: Accept an integer array arr, and integer variables low and high as parameters

Step 2: mid <-- ( low + high ) / 2

Step 3: len1 <-- mid – low + 1, len2 <-- high – mid

Step 4: Initialize an integer array left from arr[ low ... mid ]

Step 5: Initialize an integer array right from arr[ mid + 1 ... high ]

Step 6: i <--0, j <-- 0, k <-- low

Step 7: while i < len1 AND j < len2

1. if left[ i ] < right [ j ]
   1. arr [ k ] <-- left[ i ]
   2. k<--k+1, i <-- i + 1
2. else
   1. arr [ k ] <-- right [ j ]
   2. k<-k+1, j<--j + 1

Stop

**Code:**

#include <iostream>

#include <vector>

#include <algorithm>

#include <iterator>

#include <tuple>

template<typename T, typename Comp>

void merge(std::vector<T> &arr, int low, int high, Comp cmp)

{

int mid = low + (high - low) / 2;

auto[ len1, len2 ] = std::make\_pair(mid - low + 1, high - mid);

std::vector<T> left(arr.begin() + low, arr.begin() + mid + 1);

std::vector<T> right(arr.begin() + mid + 1, arr.begin() + high + 1);

auto [i, j, k] = std::make\_tuple(0, 0, low);

while(i < len1 && j < len2)

if(cmp(left[i], right[j]))

arr[k++] = left[i++];

else

arr[k++] = right[j++];

while(i < len1)

arr[k++] = left[i++];

while(j < len2)

arr[k++] = right[j++];

}

template<typename T, typename Comp>

void sort(std::vector<T> &arr, int low, int high, Comp cmp)

{

if(low < high)

{

int mid = low + (high - low) / 2;

sort(arr, low, mid, cmp);

sort(arr, mid+1, high, cmp);

merge(arr, low, high, cmp);

}

}

int main()

{

std::vector<int> arr = {std::istream\_iterator<int>(std::cin), std::istream\_iterator<int>()};

sort<int>(arr, 0, arr.size() - 1, [](int x, int y){return x<y;});

std::cout<<"Sorted array:\n";

std::for\_each(arr.begin(), arr.end(), [](const int &x){std::cout<< x << '\n';});

}

**Input:** 1 6 2 5 3 4

**Output:** Sorted array:

1

2

3

4

5

6

**Question 3:** Write a program to find the 2nd largest and 2nd smallest number in an array in O(N) time complexity using divide and conquer paradigm

**Algorithm:**

Algorithm for method main():

Start

Step 1: Enter the numbers and store it in an integer array arr.

Step 2: Create a copy of arr in arr1.

Step 3: Call subroutine sort() with arr, 0, arr.size() - 1 as parameters and less than comparator

Step 4: Print arr[ 1 ] which is the 2nd smallest element

Step 5: Call subroutine sort() with arr, 0, arr.size() - 1 as parameters and than comparator

Stop

Algorithm for subroutine sort():

Start

Step 1: Accept an integer array arr and integer variables low and high as parameters

Step 2: if low < high

1. mid <-- ( low + high ) / 2
2. Call subroutine sort() with arr, low, mid as parameters
3. Call subroutine sort() with arr, mid + 1, high as parameters
4. Call subroutine merge() with arr, low, high as parameters

Stop

Algorithm for subroutine merge():

Start

Step 1: Accept an integer array arr, and integer variables low and high as parameters

Step 2: mid <-- ( low + high ) / 2

Step 3: len1 <-- MIN(2, mid – low + 1) , len2 <-- MIN(2, high – mid)

Step 4: Initialize an integer array left from arr[ low ... low+len1-1 ]

Step 5: Initialize an integer array right from arr[ mid + 1 ... mid+len2 ]

Step 6: i <--0, j <-- 0, k <-- low

Step 7: while i < len1 AND j < len2

1. if left[ i ] < right [ j ]
   1. arr [ k ] <-- left[ i ]
   2. k<--k+1, i <-- i + 1
2. else
   1. arr [ k ] <-- right [ j ]
   2. k<-k+1, j<--j + 1

Stop

**Code:**

#include <iostream>

#include <vector>

#include <algorithm>

#include <iterator>

#include <tuple>

template<typename T, int N, typename Comp>

void merge(std::vector<T> &arr, int low, int mid, int high, Comp cmp)

{

mid = low + (high - low) / 2;

auto [len1, len2] = std::make\_pair(std::min(N, mid - low + 1), std::min(N, high - mid));

std::vector<T> left(arr.begin() + low, arr.begin() + low + len1);

std::vector<T> right(arr.begin() + mid + 1, arr.begin() + mid + 1 + len2);

auto [i, j, k] = std::make\_tuple(0, 0, low);

while(i < len1 && j < len2)

if(cmp(left[i], right[j]))

arr[k++] = left[i++];

else

arr[k++] = right[j++];

while(i < len1)

arr[k++] = left[i++];

while(j < len2)

arr[k++] = right[j++];

}

template<typename T, int N, typename Comp>

void sort(std::vector<T> &arr, int low, int high, Comp cmp)

{

if(low < high)

{

int mid = low + (high - low) / 2;

sort<T, N>(arr, low, mid, cmp);

sort<T, N>(arr, mid+1, high, cmp);

merge<T, N>(arr, low, mid, high, cmp);

}

}

int main()

{

std::vector<int> arr = {std::istream\_iterator<int>(std::cin), std::istream\_iterator<int>()};

auto arr1 = arr;

sort<int, 2>(arr, 0, arr.size() - 1, [](int x, int y){return x<y;});

std::cout<<"2nd smallest element is: "<<arr[1]<<'\n';

sort<int, 2>(arr1, 0, arr1.size() - 1, [](int x, int y){return x>y;});

std::cout<<"2nd largest element is: "<<arr1[1]<<'\n';

}

**Input:** 1 6 4 2 5 3

**Output:**

2nd smallest element is: 2

2nd largest element is: 5

**Question 4:** Given a sorted array and a number X, search two elements of array such that their sum is X. Expected time complexity is O(n).

**Algorithm:**

Start

Step 1: Enter an integer and store it in n.

Step 2: Declare and initialise an array check of size 1000 and initialise it with *false* and an array arr of size n.

Step 3: Run a loop for i = 0 to n-1

1. Input arr[i]
2. check[arr[i]] <-- *true*

Step 4: Input an integer and store it in an integer variable x and initialise a bool variable flag with false.

Step 5: Run a loop for i = 0 to n-1

1. if arr[i] <= x
   1. Initialise an integer variable y with x – arr[i]
   2. if check[y] = *true*
      1. Display the values of arr[i] and y as a pair
      2. flag <-- true
      3. break from the loop

Step 6: if flag = false

1. Print “No such pair found!!!”

Stop

**Code:**

#include <vector>

#include <iostream>

int main()

{

int n; std::cin>>n;

std::vector<bool> check(1000, false);

std::vector<int> arr(n);

for(int i = 0; i < n; i++)

{

std::cin>>arr[i];

check[arr[i]] = true;

}

int x; std::cin>>x; bool flag = false;

for(int i = 0; i < n; i++)

{

if(arr[i] <= x)

{

int y = x - arr[i];

if(check[y])

{

std::cout<<arr[i]<<","<<y<<'\n';

flag = true; break;

}

}

}

if(!flag)

std::cout<<"No such pair found!!!\n";

}

**Input:** 1 3 4 6 9

12

**Output:** 3,9

**Question 5:** Apply Binary Search on 2D NxM array (A) having numbers stored in non-deceasing order under row-major scanning. Hint: k-th element = A[k/M][k % M] for A[1…N][1…M]

**Algorithm:**

Start

Step 1: Accept two numbers and store them in integer variables m and n.

Step 2: Declare a 2D array arr of size and enter elements in arr.

Step 3: Accept the element to be searched and store it in an integer variable x.

Step 4: Initialise integer variables low, high and flag with 0, , *false*

Step5: while low <= high

1. mid <--
2. if arr[mid / n] [ mid % n ] = x
   1. flag <-- true
   2. Print the position coordinates which is (mid / n ) + 1, (mid % n) + 1
   3. break from the loop
3. else if x < arr[mid / n] [ mid % n ]
   1. high <-- mid – 1
4. else
   1. low <-- mid + 1

Step 6: If flag = false

1. Print “Element not found”

Stop

**Code:**

#include <iostream>

#include <vector>

#include <tuple>

int main()

{

int m, n; std::cin>>m>>n;

std::vector<std::vector<int>> arr(m, std::vector<int>(n));

for(auto& i:arr)

for(auto& j:i)

std::cin>>j;

std::cout<<"Enter the element to be searched\n";

int x; std::cin>>x;

auto[low, high, flag] = std::make\_tuple(0, m\*n-1, false);

while(low <= high)

{

auto mid = low + (high - low) / 2;

if(arr[mid/n][mid%n] == x)

{

flag = true;

std::cout<<"Element found at position: "<<(mid/n)+1<<","<<(mid%n)+1<<'\n';

break;

}

else if(x < arr[mid/n][mid%n])

high = mid - 1;

else

low = mid + 1;

}

if(!flag)

std::cout<<"Element not found!!!\n";

}

**Input:** 2 3

1 3 4

6 7 8

Enter the element to be searched

7

**Output:** Element found at position: 2,2

**Question 6:** Given a sorted array and a number x, write a function that counts the occurrences of x in the array. Expected time complexity is O(logn).

**Algorithm:**

Start

Algorithm for method main():

Start

Step 1: Accept an integer and store it in an integer variable n.

Step 2: Declare an integer array arr of length n and take input in it

Step 3: Accept the key to be searched and store it in an integer variable x

Step 4: Call subroutine find() twice; once with arr, x, less-than comparator and another time with arr, x, greater-than comparator and store the results in a1 and a2

Step 5: if a1 = -1 OR a2 = -1

1. Print “Not found”

Else

1. Print a2 – a1 + 1

Stop

Algorithm for subroutine find():

Start

Step 1: Accept an integer array arr, an integer key and a comparator cmp as parameters

Step 2: Initialize integer variables low, high, mid and ans with 0, length(arr) – 1, 0, -1

Step 3: while ( low <= high )

1. mid <-- ( low + high ) / 2
2. if key = arr[ mid ]
   1. ans <-- mid
   2. if cmp(1, 2) = *true*
      * 1. high <-- mid – 1

else

* + - 1. low <-- mid + 1

else if cmp( key, arr[ mid ] = *true*)

* + 1. if cmp(1, 2) = *true*
       - 1. high <-- mid – 1

Else

* + - * 1. low <-- mid + 1

Else

* + 1. if cmp(1, 2) = *true*
       - 1. low <-- mid + 1

else

* + - * 1. high <-- mid – 1

Step 4: Return ans

Stop

Stop

**Code:**

#include <iostream>

#include <vector>

#include <tuple>

#include <algorithm>

template<typename Cmp>

int find(std::vector<int> &arr, int &key, Cmp cmp)

{

auto[low, high, mid, ans] = std::make\_tuple(0, arr.size()-1, 0, -1);

while(low <= high)

{

mid = low + (high - low) / 2;

if(key == arr[mid])

{

ans = mid;

if(cmp(1, 2))

high = mid - 1;

else

low = mid + 1;

}

else if(cmp(key, arr[mid]))

{

if(cmp(1, 2))

high = mid - 1;

else

low = mid + 1;

}

else

{

if(cmp(1, 2))

low = mid + 1;

else

high = mid - 1;

}

}

return ans;

}

int main()

{

int n; std::cin>>n;

std::vector<int> arr(n);

for(auto& i:arr)

std::cin>>i;

std::cout<<"Enter the key to be searched\n";

int x; std::cin>>x;

auto[a1, a2] = std::make\_tuple(find(arr, x, std::less<int>()), find(arr, x, std::greater<int>()));

if(a1 == -1 || a2 == -1)

std::cout<<"Not found!!!\n";

else

std::cout<<a2 - a1 + 1<<'\n';

}

**Input:** 8

1 3 3 3 5 6 6 9

Enter the key to be searched

3

**Output:** 3

**Question 7:** You are given an array that represents elements of arithmetic progression in order. One element is missing in the progression. Find the missing number.

**Algorithm:**

Start

Algorithm for method main():

Start

Step 1: Accept an integer and store it in an integer variable n.

Step 2: Declare an integer array arr of size n and accept elements in it.

Step 3: Print the returned value by invoking find( arr, 0, n-1, (arr[n-1] – arr[0]) / n )

Stop

Algorithm for subroutine find():

Start

Step 1: Accept an integer array arr and integer variables low, high and diff as parameters

Step 2: Initialise an integer variable mid with ( low + high ) / 2

Step 3: If

return arr[ mid ] + diff

Step 4: If AND

return arr[ mid - 1 ] + diff

Step 5: arr[ mid ] = arr[ 0 ] +

return find(arr, mid + 1, high, diff)

Step 6: return find( arr, low, mid – 1, diff )

Stop

Stop

**Code:**

#include <iostream>

#include <vector>

int find(std::vector<int> &arr, int low, int high, int diff)

{

int mid = (low + high) / 2;

if((arr[mid+1] - arr[mid]) != diff)

return arr[mid]+diff;

if(mid != 0 && arr[mid] - arr[mid-1] != diff)

return arr[mid-1]+diff;

if(arr[mid] == arr[0] + diff\*mid)

return find(arr, mid+1, high, diff);

return find(arr, low, mid-1, diff);

}

int main()

{

int n; std::cin>>n;

std::vector<int> arr(n);

for(auto& i:arr)

std::cin>>i;

std::cout<<find(arr, 0, n-1, (arr[n-1] - arr[0])/n)<<'\n';

}

**Input:** 9

1 3 7 9 11 13 15 17 19

**Output:** 5

**Question 8:** A Bitonic Sequence is a sequence of numbers which is first strictly increasing then after a point strictly decreasing. A Bitonic Point is a point in bitonic sequence before which elements are strictly increasing and after which elements are strictly decreasing. Find bitonic point in a bitonic sequence.

**Algorithm:**

Start

Algorithm for method main():

Start

Step 1: Accept an integer and store it in an integer variable n.

Step 2: Declare an integer array of size n and accept elements in it.

Step 3: Print the result returned from calling find( arr, 0, n-1 )

Stop

Algorithm for subroutine find():

Start

Step 1: Accept an integer array arr and two integer variables low and high

Step 2: Initialise an integer variable mid with ( low + high ) / 2

Step 3: if low <= high

1. if arr[ mid ] > arr[ mid - 1 ] AND arr[ mid ] > arr[ mid + 1 ]
   1. return mid

else if arr[ mid ] < arr[ mid + 1 ]

* 1. return find(arr, mid + 1, high)

else

* 1. return find(arr, low, mid – 1)

Step 4: Return mid

Stop

Stop

**Code:**

#include <iostream>

#include <vector>

int find(std::vector<int> &arr, int low, int high)

{

int mid = (high + low)/2;

if(low <= high)

{

if(arr[mid] > arr[mid-1] && arr[mid] > arr[mid+1])

return mid;

else if(arr[mid] < arr[mid+1])

return find(arr, mid+1, high);

else

return find(arr, low, mid-1);

}

return mid;

}

int main()

{

int n; std::cin>>n;

std::vector<int> arr(n);

for(auto& i:arr)

std::cin>>i;

std::cout<<find(arr, 0, n-1)<<'\n';

}

**Input:** 7

10 20 30 40 20 -10 -100

**Output:** 3