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