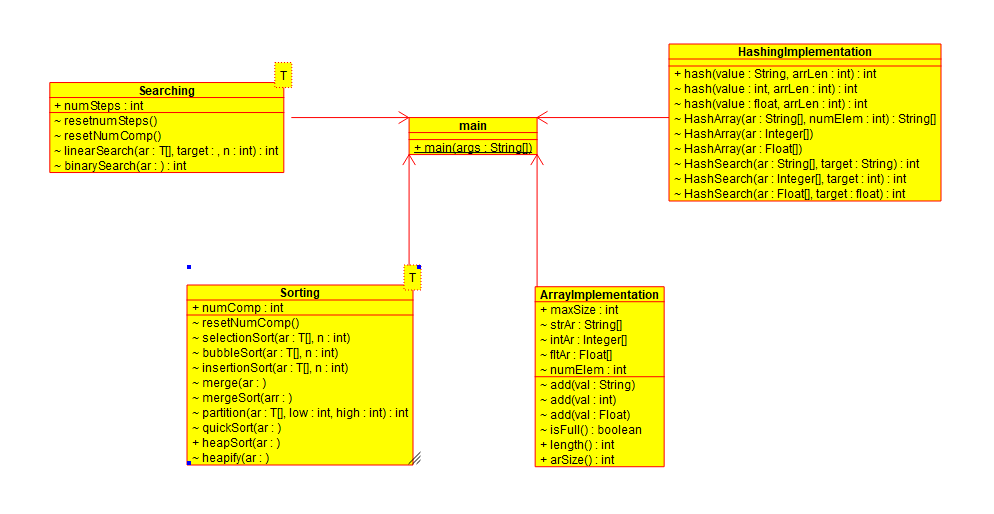
**UML DIAGRAM:**

****

**PSEUDO CODES:**

**The mainly used algorithms are:**

**1)Selection Sort:**

void selectionSort(T[] ar,int n) :

for int i = 0 i < n-1

i++

int minInd = i

for int j = i+1 j < n j++

if ar[j]<ar[minInd]

minInd = j

swap ar[j],ar[minInd]

**2)Bubble Sort**

void bubbleSort(T[] ar,int n)

**for** **int** i=0 i<n-1 i++

**int** flag=0

**for**(**int** j=0j<n-i-1j++)

**if** ar[j]>ar[j+1]

swap ar[j],ar[j+1]

flag=1

**if** (flag==0)

**break**

**3)Insertion Sort**

**for**(**int** i=1i<n++i)

T value=ar[i]

**int** node=i-1

**while**(node>=0 and ar[node].compareTo(value)>0)

numComp++

ar[node+1]=ar[node]

node=node-1

ar[node+1]=value

**4)Merge Sort**

**void** mergeSort(T arr[], **int** left, **int** right)

**if** left < right

**int** mid = left + (right - left) / 2

mergeSort(arr, left, mid

mergeSort(arr, mid + 1,right)

merge(arr, left, mid, right)

**Merge Method**

**void** merge(T ar[], **int** left, **int** mid, **int** right

**int** n1 = mid - left + 1

**int** n2 = right - mid

T[] leftArr = (T[]) **new** Comparable[n1]

T[] rightArr = (T[])**new** Comparable[n2]

**for** (**int** i = 0 i < n1 ++i) {

leftArr[i] = ar[left + i]

**for** (**int** j = 0 j < n2 ++j)

rightArr[j] = ar[mid + 1 + j

**int** i = 0

**int** j = 0

**int** k = left

**while** (i < n1 && j < n2)

**if** (leftArr[i]< rightArr[j]

ar[k] = leftArr[i

i++

**else**

ar[k] = rightArr[j

j++

k++

**while** (i < n1)

ar[k] = leftArr[i]

i++

k++

**while** (j < n2) {

ar[k] = rightArr[j]

j++

k++

**5)Quick Sort**

**void** quickSort(T ar[], **int** low, **int** high)

**if** (low < high)

**int** part = partition(ar, low, high)

quickSort(ar, low, part - 1)

quickSort(ar, part + 1, high)

**Partition:**

**int** partition(T[] ar, **int** low, **int** high)

T pivot = ar[high]

**int** i = low - 1//index of smaller element

**for** (**int** j = low j <= high - 1 j++) {

**if** (ar[j].compareTo(pivot)<0)

i++

swap ar[i],ar[j]

swap ar[i+1],ar[high]

**return** index

**6)Heap Sort**

**public** **void** heapSort(T ar[], **int** n)

**for** (**int** i = n / 2 – 1 i >= 0 i--)

heapify(ar, n, i)

**for** (**int** i = n - 1 i > 0 i--) {

swap ar[0],ar[i]

heapify(ar, i, 0)

**Heapify:**

**void** heapify(T ar[], **int** n, **int** i)

**int** max = i

**int** left = 2 \* i + 1

**int** right = 2 \* i + 2

**if** (left < n and ar[left]>ar[maximum]

max = left

**if** (right < n and ar[right]> ar[max]

max = right

numComp++

**if** (max != i)

T temp = ar[i]

ar[i] = ar[max]

ar[max] = temp

heapify(ar, n, max)

**7)Linear Search**

**int** linearSearch(T[] ar,T target,**int** n

**for**(**int** i=0i<ni++) {

**if**(ar[i]==target)

**return** index

else -1

**8)Binary Search**

**int** binarySearch(T ar[], **int** left, **int** right, T target)

**if** (right >= left)

**int** mid = left + (right - left) / 2

**if** (ar[mid].compareTo(target)==0)

**return** mid

**if** (ar[mid].compareTo(target)>0)

**return** binarySearch(ar, left, mid - 1, target)

**else**

**return** binarySearch(ar, mid + 1, right, target

**return** -1

**9)Hash Search:**

**boolean** found=**false**

**int** pos=0

**int** index=hash(target,ar.length)

**int** curIndex = (index+1)%ar.length

**if**(ar[index]!=**null** and ar[index]==target)

pos= index

found=**true**

numSteps++

curIndex++

**while**(ar[curIndex]!=**null** and curIndex!=index ) **if**(ar[curIndex]==target)

pos = curIndex

found=**true**

curIndex=(curIndex+1)%ar.length

**if**(found==**true**)

**return** position

**else**

**return** -1