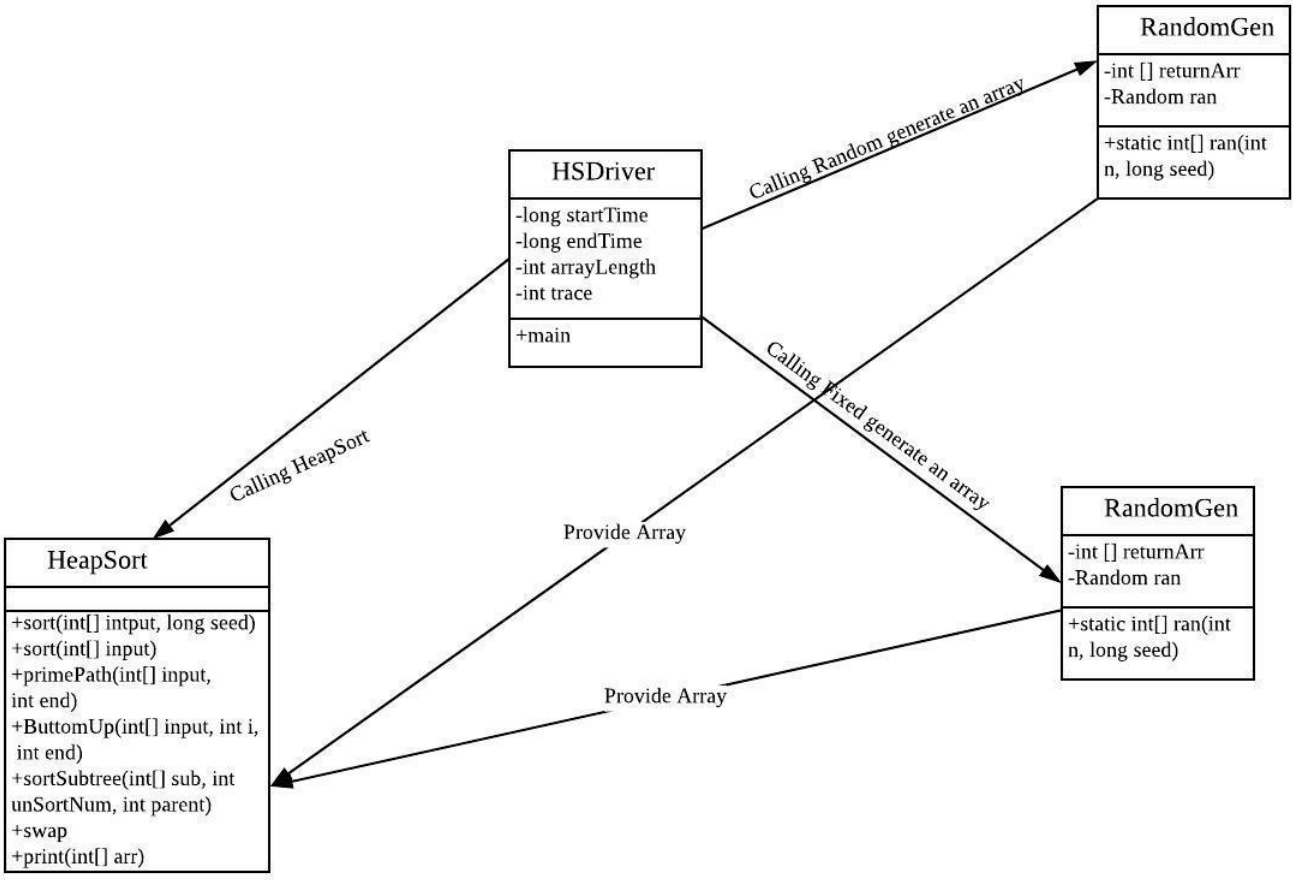
Question 1

(a) class diagram



(b)

For better reading experience, we conclude picture as follow. For text form pseudocode please check the end of pdf

```
Algorithm HeapSort()
Input: unsorted Array, traceStep
Output: sorted array
Print Unsorted Array
for (length/2-1 to 0)  //build max heap
sortSubtree(inputArray,length,i)
if(trace=0)
print arrary;
for (length/2-1 to 0){  //re-settle each
  element and put [0] to sorted part
  swap(input[i],input[0])
  ButtomUp(inputArray,i,i)
  if(trace=i)
  print array
}
print sorted array
```

```
Algorithm ButtomUp()
prime←primePath(input,end)

while(input[i]>input[prime]){
  prime←prime's parent
} //find the valid position to insert
temp←input[prime]
input[prime]←input[i]
while(prime>0){
  swap(temp,input[prime])
  prime←prime's parent
} //insert [i] and move prime 1 position up
```

```
Algorithm PrimePath()
input: array, end
output: primeIndex
prime←0
while(prime's child<=end){
  if(prime's rightChild>=leftChild)
  prime=rightChild
  else
  prime=leftChild
}
return prime
```

```
Algorithm sortSubtree{
Input:array, unSortNumber, parentIndex;
Output: sorted subTree;
greatest←parent;
left←2*parent+1;
right←2*parent+1;
if(left>greatest or right>greatest)
greatest←left or greatet←right;
if(greatest!=parent){
  swap(greatest,parent);
  sortSubtree(inputArrary, i, greatest);
}
}

Algorithm HSDriver()
if (type==RandomGen){
unSortedArr←RandomGen.ran}
else{
unSortedArr←FixedGen.ran}
if(trace<0){
  HeapSort without trace
}
else{
  HeapSort with trace
}
Print(System end-start runtime)
```

(c)

For heap sort class. It accepts an array and first builds it into a max heap tree by calling subTree() using bottom-up construction. After building, for each element in the array, it first finds the prime path of unsorted part array using PrimePath(), then compares a[end] element with each element of the path and find the valid insert position using BottomUp(). After finding the position, it shifts a[0] to sorted part, shifts each primePath element up 1 position until root, and shift a[end] to the valid insert position. Keep doing so for each element until this is a sorted array.

Question 2

Size	Run Time
10	1926371
50	3662868
100	5259623
5000	79626615
10000	131088348
1000000	6210561802

Question 3

Construction: top-down starting from array's beginning then insert and shift. Total takes nlogn times. Bottom-up start from the array's end, forms a tree then shift leaves. It total takes n times.
Sorting: top-down puts a[0] largest element into sorted array, then swap a[0] and a[end] , shift new element down. New element comes from leaf, it's smaller and needs more compare. But bottom-up sort find the prime path that all greater than siblings and therefore only need 1 compare each level. Bottom-up sort need average $\mathrm{logn} + \mathrm{O(n)}$ compare, but top-down need average $2\mathrm{logn} + \mathrm{O(n)}$ compare. Therefore bottom-up is more efficient.

Question 4

First use primePath() method find the prime path. Its elements are bigger than siblings, so only need 1 compare each, there are total logn levels compare. Then use ButtomUp() method to compare a[end] with all prime path elements to find the valid position to insert a[end] element, once we found it, shift current a[0] to sorted array part, shift all other prime path elements up one position, and insert a[end] to valid position. Keeping doing so until all elements are sorted. That is, constructing a heap takes O(n), re-settle each element take 1logn time, total n element, it takes $\mathrm{logn} + \mathrm{O(n)}$ average time.

Pseudocode

```
if (type==RandomGen){
unSortedArr←RandomGen.ran}
else{
unSortedArr←FixedGen.ran}
if(trace<0){
   HeapSort without trace
}
else{
   HeapSort with trace
}
Print(System end-start runtime)

Algorithm ButtomUp()
prime←primePath(input,end)

while(input[i]>input[prime]){
   prime←prime's parent
} //find the valid position to insert
temp←input[prime]
input[prime]←input[i]
while(prime>0){
   swap(temp,input[prime])
   prime←prime's parent
} //insert [i] and move prime 1 position up

Algorithm PrimePath()
input: array, end
output: primeIndex
prime←0
while(prime's child<=end){
   if(prime's rightChild>=leftChild)
   prime=rightChild
   else
   prime=leftChild
}
return prime


Algorithm HeapSort()
Input: unsorted Array, traceStep
Output: sorted array
Print Unsorted Array
```

```
for (length/2-1 to 0)    //build max heap
sortSubtree(inputArray,length,i)
if(trace=0)
print arrary;
for (length/2-1 to 0){    //re-settle each element and put [0] to sorted part
   swap(input[i],input[0])
   ButtomUp(inputArray,i,i)
   if(trace=i)
   print array
}
print sorted array


Algorithm sortSubtree{
Input:array, unSortNumber, parentIndex;
Output: sorted subTree;
greatest←parent;
left←2*parent+1;
right←2*parent+1;
if(left>greatest or right>greatest)
greatest←left or greatet←right;
if(greatest!=parent){
   swap(greatest,parent);
   sortSubtree(inputArrary, i, greatest);
}
}
```