ITCS 6114: DATA STRUCTURES AND ALGORITHM

Project 1

Student Name: Nisha Ramrakhyani (Id: 801208678)

REPORT SUMMARY & DATA STRUCTURES USED:

This Report contains complexity analysis of following algorithms based on the number of inputs.

a) Insertion Sort (Using an array)
b) Merge Sort(Using an array)
c) Vector based Heap Sort(Using a vector and array)
d) In-place Quick Sort(Using an array)
e) Modified Quick Sort(Using an array)

Time taken for each algorithm is computed by the function timeit.default_timer(). Also, each algorithm is given random input, sorted input and reverse sorted input to test the complexity of each algorithm.

COMPLEXITY ANALYSIS AND RESULTS:

Algorithm	Average Case	Best Case	Worst Case
Insertion Sort	$O(n^2)$	O(n)	$O(n^2)$
Merge Sort	O(n log (n))	O(n log (n))	O(n log (n))
Heap Sort	O(n log (n))	O(n log (n))	O(n log (n))
In-place Quick Sort	O(n log (n))	O(n log (n))	$O(n^2)$
Modified Quick Sort	O(n log (n))	O(n log (n))	$O(n^2)$

As we can see in the graphs below,

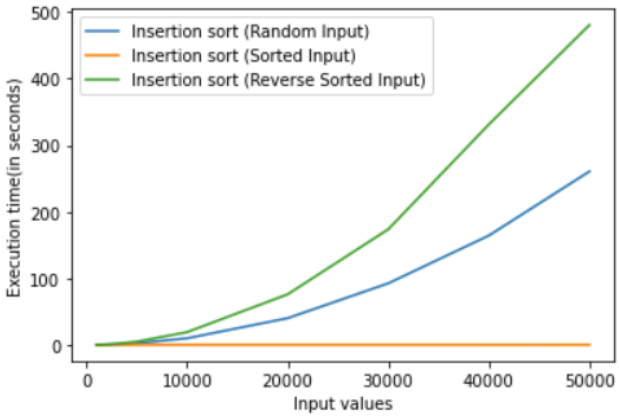
a) Insertion sort works best if the input array is already sorted and if the dataset is small. Best case complexity of Insertion sort is O(n). However, if the array is reverse sorted, the complexity of Insertion sort becomes $O(n^2)$ because of the number of swaps between elements. Also, for random input array, complexity for Insertion sort is $O(n^2)$.
b) Merge Sort works best for large number of inputs. Complexity of merge sort is same for all the cases because it is independent of distribution of the data. However, Merge sort is not an in-place algorithm as it requires extra space to merge the sorted subarrays.
c) Quicksort works best for the random input. For a sorted/reverse sorted input array, the complexity of Quicksort becomes $O(n^2)$. Quicksort is faster than merge sort because there is no extra juggling as in mergesort.
d) Heap Sort also has the same complexity for all the cases. Heap Sort is suitable for input size less than 1M.

e) Heap Sort and Merge sort are comparatively faster algorithms. However, Merge sort is the most suitable algorithm for huge datasets(>1M).

Simulation(Instruction 1)-Output and figure

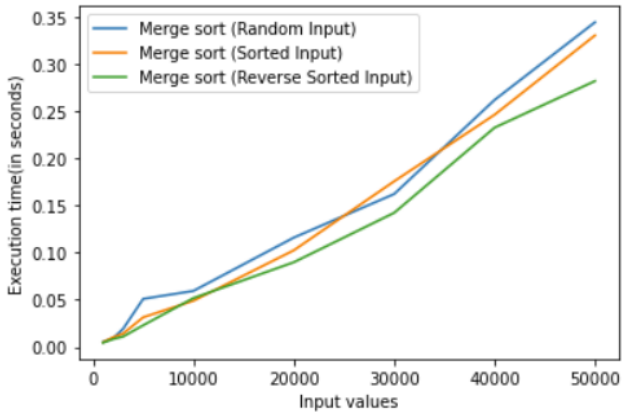
COMPARISON PLOT FOR INSERTION SORT:

```
#Comparison plot for Insertion Sort
plt.plot(val,Execution_time_arr,label='Insertion sort (Random Input)')
plt.plot(val,Execution_time_arr1,label='Insertion sort (Sorted Input)')
plt.plot(val,Execution_time_arr2,label='Insertion sort (Reverse Sorted Input)')
plt.xlabel('Input values')
plt.ylabel('Execution time(in seconds)')
plt.legend()
plt.show()
```



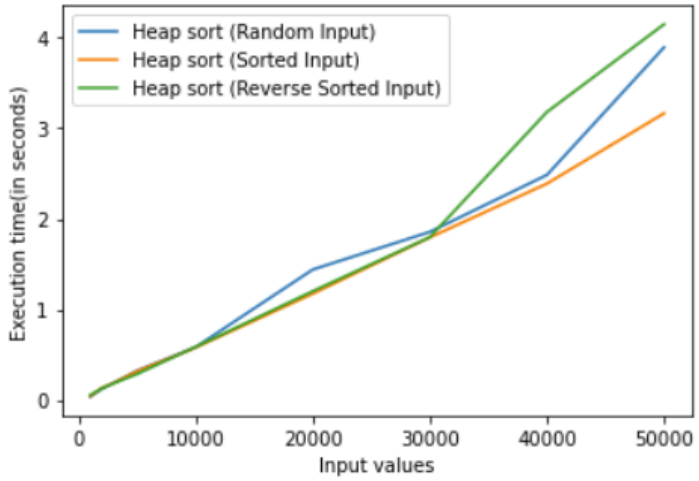
COMPARISON PLOT FOR MERGE SORT:

```
#Comparison plot for Merge Sort
plt.plot(val,Execution_time_arr3,label='Merge sort (Random Input)')
plt.plot(val,Execution_time_arr4,label='Merge sort (Sorted Input)')
plt.plot(val,Execution_time_arr5,label='Merge sort (Reverse Sorted Input)')
plt.xlabel('Input values')
plt.ylabel('Execution time(in seconds)')
plt.legend()
plt.show()
```



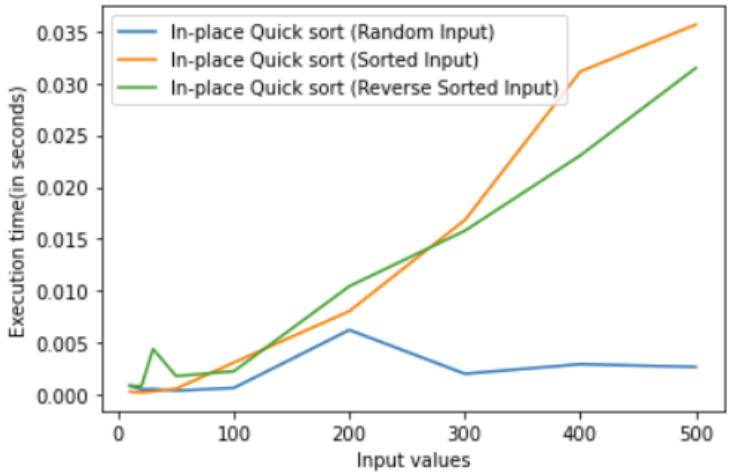
COMPARISON PLOT FOR HEAP SORT:

```
#Comparison plot for Heap Sort
plt.plot(val,Execution_time_arr6,label='Heap sort (Random Input)')
plt.plot(val,Execution_time_arr7,label='Heap sort (Sorted Input)')
plt.plot(val,Execution_time_arr8,label='Heap sort (Reverse Sorted Input)')
plt.xlabel('Input values')
plt.ylabel('Execution time(in seconds)')
plt.legend()
plt.show()
```



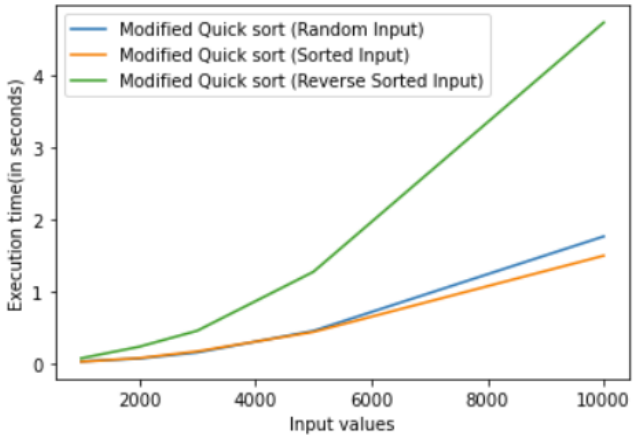
COMPARISON PLOT FOR IN_PLACE QUICK SORT:

```
#Comparison plot for In-place Quick Sort
plt.plot(val,Execution_time_arr9,label='In-place Quick sort (Random Input)')
plt.plot(val,Execution_time_arr10,label='In-place Quick sort (Sorted Input)')
plt.plot(val,Execution_time_arr11,label='In-place Quick sort (Reverse Sorted Input)')
plt.xlabel('Input values')
plt.ylabel('Execution time(in seconds)')
plt.legend()
plt.show()
```



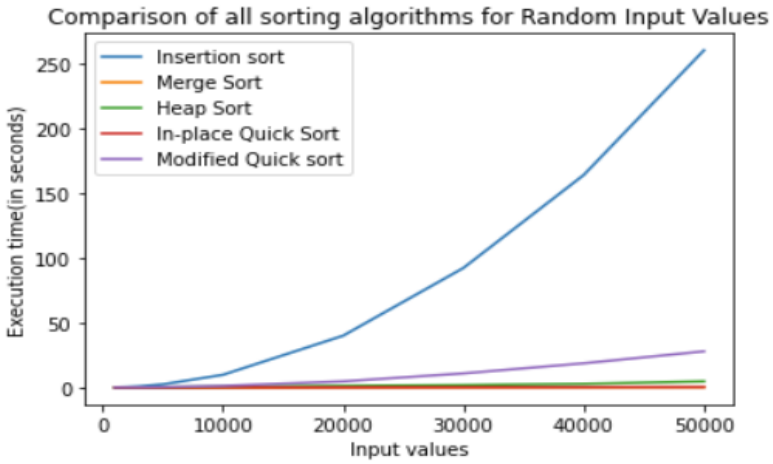
COMPARISON PLOT FOR MODIFIED QUICKSORT:

```
#Comparison plot for Modified Quick Sort
plt.plot(val,Execution_time_arr12,label='Modified Quick sort (Random Input)')
plt.plot(val,Execution_time_arr13,label='Modified Quick sort (Sorted Input)')
plt.plot(val,Execution_time_arr14,label='Modified Quick sort (Reverse Sorted Input)')
plt.xlabel('Input values')
plt.ylabel('Execution time(in seconds)')
plt.legend()
plt.show()
```



COMPARISON OF ALL ALGORITHMS FOR RANDOM INPUT:

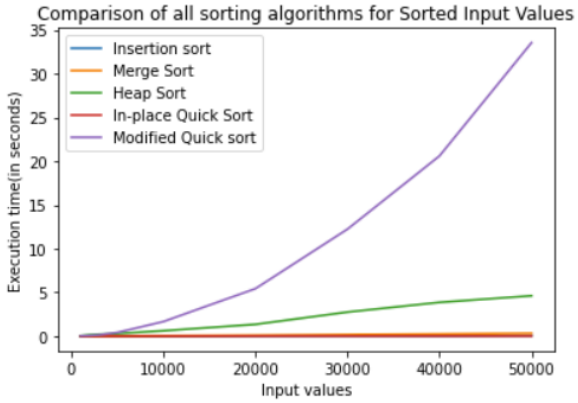
```
#Comparison of all sorting algorithms for Random Input
plt.plot(val,Execution_time_arr,label='Insertion sort')
plt.plot(val,Execution_time_arr3,label='Merge Sort')
plt.plot(val,Execution_time_arr6,label='Heap Sort')
plt.plot(val,Execution_time_arr9,label='In-place Quick Sort')
plt.plot(val,Execution_time_arr12,label='Modified Quick sort')
plt.title('Comparison of all sorting algorithms for Random Input Values')
plt.xlabel('Input values')
plt.ylabel('Execution time(in seconds)')
plt.legend()
plt.show()
```



Special cases(Instruction 2)

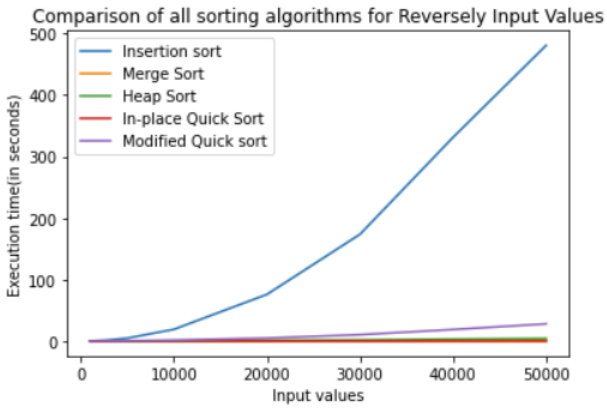
COMPARISON OF ALL ALGORITHMS FOR SORTED INPUT:

```
#Comparison of all sorting algorithms for Sorted Input
plt.plot(val,Execution_time_arr1,label='Insertion sort')
plt.plot(val,Execution_time_arr4,label='Merge Sort')
plt.plot(val,Execution_time_arr7,label='Heap Sort')
plt.plot(val,Execution_time_arr10,label='In-place Quick Sort')
plt.plot(val,Execution_time_arr13,label='Modified Quick sort')
plt.title('Comparison of all sorting algorithms for Sorted Input Values')
plt.xlabel('Input values')
plt.ylabel('Execution time(in seconds)')
plt.legend()
plt.show()
```



COMPARISON OF ALL ALGORITHMS FOR REVERSE SORTED INPUT:

```
#Comparison of all sorting algorithms for Reversely Sorted Input
plt.plot(val,Execution_time_arr2,label='Insertion sort')
plt.plot(val,Execution_time_arr5,label='Merge Sort')
plt.plot(val,Execution_time_arr8,label='Heap Sort')
plt.plot(val,Execution_time_arr11,label='In-place Quick Sort')
plt.plot(val,Execution_time_arr14,label='Modified Quick sort')
plt.title('Comparison of all sorting algorithms for Reversely Input Values')
plt.xlabel('Input values')
plt.ylabel('Execution time(in seconds)')
plt.legend()
plt.show()
```



CODE:

```
from time import process_time
import random
import sys
import math
import timeit
import matplotlib.pyplot as plt
val = [1000,2000,3000,5000,10000,20000,30000,40000,50000]

#Insertion Sort (With Random input values)

Execution_time_arr = []
for j in val:
   insertion_sort_arr = []
   random.seed(3)
   # Generating random numbers
   insertion_sort_arr = random.sample(range(0,50000),j)
   start = None
   #Keeping track of the time using timeit.default_timer() function
   start = timeit.default_timer()
   print("\nGiven array:",insertion_sort_arr)
   def insertion_sort(s):
      for i in range(1, len(s)):
         key = s[i]
         j = i - 1
         while j >= 0 and key < s[j]:
            s[j + 1] = s[j]
```

```
            j -= 1
        s[j + 1] = key

    print('\nThe sorted list after insertion sort: \t', s)
    print('\n')

insertion_sort(insertion_sort_arr)
end = None
end = timeit.default_timer()

Execution_time = end - start
Execution_time_arr.append(Execution_time)
print("Execution Time in seconds:",Execution_time)

print("Execution Time of each run:",Execution_time_arr)

#Plotting the graph for Insertion sort with Random input array
plt.plot(val,Execution_time_arr,label='Insertion sort (Random Input)')
plt.xlabel('Input values')
plt.ylabel('Execution time(in seconds)')
plt.legend()
plt.show()

#If the input array for Insertion sort is already sorted

Execution_time_arr1 = []
for j in val:
```

```
insertion_sort_arr = []
random.seed(3)
insertion_sort_arr = random.sample(range(0,50000),j)
start = None
start = timeit.default_timer()
#Sorting the array
insertion_sort_arr = sorted(insertion_sort_arr)
print("\nGiven array",insertion_sort_arr)
def insertion_sort(s):
    for i in range(1, len(s)):
        key = s[i]
        j = i - 1
        while j >= 0 and key < s[j]:
            s[j + 1] = s[j]
            j -= 1
        s[j + 1] = key

    print('\nThe sorted list after insertion sort (sorted input array): \t', s)
    print('\n')

insertion_sort(insertion_sort_arr)
end = None
end = timeit.default_timer()

Execution_time = end - start
Execution_time_arr1.append(Execution_time)
print("Execution Time in seconds:",Execution_time)
```

```
print("Execution Time for each run in seconds:",Execution_time_arr1)

#Plotting the graph for Insertion sort with already sorted input array
plt.plot(val,Execution_time_arr1,label='Insertion sort (Sorted Input)')
plt.xlabel('Input values')
plt.ylabel('Execution time(in seconds)')
plt.legend()
plt.show()

#If the input array for Insertion sort is reversely sorted
Execution_time_arr2 = []
for j in val:
   insertion_sort_arr = []
   random.seed(3)
   insertion_sort_arr = random.sample(range(0,50000),j)
   start = None
   start = timeit.default_timer()
    #Sorting the array in reverse order
   insertion_sort_arr = sorted(insertion_sort_arr,reverse=True)
   print("Given array",insertion_sort_arr)
   def insertion_sort(s):
      for i in range(1, len(s)):
         key = s[i]
         j = i - 1
         while j >= 0 and key < s[j]:
            s[j + 1] = s[j]
```

```
            j -= 1
        s[j + 1] = key

    print('\nThe sorted list after insertion sort(Reversely Sorted Input): \t', s)
    print('\n')

insertion_sort(insertion_sort_arr)
end = None
end = timeit.default_timer()
Execution_time = end - start
Execution_time_arr2.append(Execution_time)
print("Execution Time in seconds:",Execution_time)

print("Execution Time for each run in seconds:",Execution_time_arr2)

#Plotting the graph for Insertion sort with reversely sorted input array
plt.plot(val,Execution_time_arr2,label='Insertion sort (Reversely Sorted Input)')
plt.xlabel('Input values')
plt.ylabel('Execution time(in seconds)')
plt.legend()
plt.show()

#Comparison plot for Insertion Sort
plt.plot(val,Execution_time_arr,label='Insertion sort (Random Input)')
plt.plot(val,Execution_time_arr1,label='Insertion sort (Sorted Input)')
plt.plot(val,Execution_time_arr2,label='Insertion sort (Reverse Sorted Input)')
plt.xlabel('Input values')
```

```
plt.ylabel('Execution time(in seconds)')
plt.legend()
plt.show()

# MergeSort (With Random input values)

Execution_time_arr3 = []
for j in val:
    merge_sort_arr = []
    random.seed(3)
    merge_sort_arr = random.sample(range(0,50000),j)
    start = None
    start = timeit.default_timer()
    print("\nGiven array: ",merge_sort_arr)
    def mergesort(input_array):
        if len(input_array) > 1:
            mid = math.floor(len(input_array) / 2)
            left = input_array[: mid]
            right = input_array[mid:]
            mergesort(left)
            mergesort(right)
            merge(left, right, input_array)
        return input_array

    def merge(left, right, input_array2):
        i = 0
        j = 0
```

```
        k = 0
        while i != len(left) or j != len(right):
            if j == len(right) or (i < len(left) and left[i] < right[j]):
                input_array2[k] = left[i]
                i = i + 1
            else:
                input_array2[k] = right[j]
                j = j + 1
            k = k + 1

    sorted_array = mergesort(merge_sort_arr)

    print("\nSorted array:",sorted_array)
    end = None
    end = timeit.default_timer()
    Execution_time=end - start
    Execution_time_arr3.append(Execution_time)
    print("\nExecution Time in seconds:\t",Execution_time)
print("\nExecution Time for each run in seconds:\t",Execution_time_arr3)

#Plotting the graph for Merge sort with random input array
plt.plot(val,Execution_time_arr3,label='Merge sort (Random Input)')
plt.xlabel('Input values')
plt.ylabel('Execution time(in seconds)')
plt.legend()
plt.show()
```

```
#If the array for Merge sort is already sorted
Execution_time_arr4 = []
for j in val:
    merge_sort_arr = []
    random.seed(3)
    merge_sort_arr = random.sample(range(0,50000),j)
    start = None
    start = timeit.default_timer()
    #Sorting the array
    merge_sort_arr = sorted(merge_sort_arr)
    print("\nGiven array: ",merge_sort_arr)
    def mergesort(input_array):
        if len(input_array) > 1:
            mid = math.floor(len(input_array) / 2)
            left = input_array[: mid]
            right = input_array[mid:]
            mergesort(left)
            mergesort(right)
            merge(left, right, input_array)
        return input_array

    def merge(left, right, input_array2):
        i = 0
        j = 0
        k = 0
        while i != len(left) or j != len(right):
            if j == len(right) or (i < len(left) and left[i] < right[j]):
```

```
            input_array2[k] = left[i]
            i = i + 1
        else:
            input_array2[k] = right[j]
            j = j + 1
        k = k + 1

    sorted_array = mergesort(merge_sort_arr)

    print("\nSorted array:",sorted_array)
    end = None
    end = timeit.default_timer()
    Execution_time=end - start
    Execution_time_arr4.append(Execution_time)
    print("\nExecution Time in seconds:\t",Execution_time)
print("\nExecution Time for each run in seconds:\t",Execution_time_arr4)

#Plotting the graph for Merge sort with sorted input array
plt.plot(val,Execution_time_arr4,label='Merge sort (Sorted Input)')
plt.xlabel('Input values')
plt.ylabel('Execution time(in seconds)')
plt.legend()
plt.show()

#If the array for Merge sort is reversely sorted
Execution_time_arr5 = []
for j in val:
```

```
merge_sort_arr = []
random.seed(3)
merge_sort_arr = random.sample(range(0,50000),j)
start = None
start = timeit.default_timer()
#Sorting the array
merge_sort_arr = sorted(merge_sort_arr,reverse=True)
print("\nGiven array: ",merge_sort_arr)
def mergesort(input_array):
    if len(input_array) > 1:
        mid = math.floor(len(input_array) / 2)
        left = input_array[: mid]
        right = input_array[mid:]
        mergesort(left)
        mergesort(right)
        merge(left, right, input_array)
    return input_array

def merge(left, right, input_array2):
    i = 0
    j = 0
    k = 0
    while i != len(left) or j != len(right):
        if j == len(right) or (i < len(left) and left[i] < right[j]):
            input_array2[k] = left[i]
            i = i + 1
        else:
```

```
            input_array2[k] = right[j]
            j = j + 1
        k = k + 1

    sorted_array = mergesort(merge_sort_arr)

    print("\nSorted array:",sorted_array)
    end = None
    end = timeit.default_timer()
    Execution_time=end - start
    Execution_time_arr5.append(Execution_time)
    print("\nExecution Time in seconds:\t",Execution_time)
print("\nExecution Time for each run in seconds:\t",Execution_time_arr5)

#Plotting the graph for Merge sort with reversely sorted input array
plt.plot(val,Execution_time_arr5,label='Merge sort (Reverse sorted Input)')
plt.xlabel('Input values')
plt.ylabel('Execution time(in seconds)')
plt.legend()
plt.show()

#Comparison plot for Merge Sort
plt.plot(val,Execution_time_arr3,label='Merge sort (Random Input)')
plt.plot(val,Execution_time_arr4,label='Merge sort (Sorted Input)')
plt.plot(val,Execution_time_arr5,label='Merge sort (Reverse Sorted Input)')
plt.xlabel('Input values')
plt.ylabel('Execution time(in seconds)')
```

```
plt.legend()
plt.show()


# vector based heapsort
Execution_time_arr6 = []
for j in val:
    heapsort_arr = []
    random.seed(3)
    heapsort_arr = random.sample(range(0,50000),j)
    start = None
    start = timeit.default_timer()
    print("\nGiven array: ",heapsort_arr)
    class BinHeap:
        #initializing the vector and an attribute currentSize as 0 to allow for interger division
        def __init__(self):
            self.heapList = [0]
            self.currentSize = 0


        #prelocating items far up in the tree to maintain heap property
        def HeapUp(self,i):
            while i // 2 > 0:
                if self.heapList[i] < self.heapList[i // 2]:
                    tmp = self.heapList[i // 2]
                    self.heapList[i // 2] = self.heapList[i]
                    self.heapList[i] = tmp
                i = i // 2
```

```
# appends item to the end of the vector
def insert(self,k):
    self.heapList.append(k)
    self.currentSize = self.currentSize + 1
    self.HeapUp(self.currentSize)

#prelocating items far down in the tree to maintain heap property
def HeapDown(self,i):
    while (i * 2) <= self.currentSize:
        mc = self.minChild(i)
        if self.heapList[i] > self.heapList[mc]:
            tmp = self.heapList[i]
            self.heapList[i] = self.heapList[mc]
            self.heapList[mc] = tmp
        i = mc

def minChild(self,i):
    if i * 2 + 1 > self.currentSize:
        return i * 2
    else:
        if self.heapList[i*2] < self.heapList[i*2+1]:
            return i * 2
        else:
            return i * 2 + 1

def delMin(self):
    retval = self.heapList[1]
```

```
        self.heapList[1] = self.heapList[self.currentSize]
        self.currentSize = self.currentSize - 1
        self.heapList.pop()
        self.HeapDown(1)
        return retval

    def buildHeap(self,alist):
        i = len(alist) // 2
        self.currentSize = len(alist)
        self.heapList = [0] + alist[:]
        while (i > 0):
            self.HeapDown(i)
            i = i - 1

def main():

    bh = BinHeap()
    bh.buildHeap(heapsort_arr)

    for i in range(len(heapsort_arr)):
        print(bh.delMin())

main()
end = None
end=timeit.default_timer()
Execution_time=end - start
Execution_time_arr6.append(Execution_time)
```

```
    print("\nExecution Time in seconds:",Execution_time)

print("\nExecution Time for each run in seconds:\t",Execution_time_arr6)



#Plotting the graph for Heap sort with random input array
plt.plot(val,Execution_time_arr6,label='Heap sort (Random Input)')
plt.xlabel('Input values')
plt.ylabel('Execution time(in seconds)')
plt.legend()
plt.show()

#If the array for heap sort is already sorted
Execution_time_arr7 = []
for j in val:
    heapsort_arr = []
    random.seed(3)
    heapsort_arr = random.sample(range(0,50000),j)
    start = None
    start = timeit.default_timer()
    #Sorting the array
    heapsort_arr = sorted(heapsort_arr)
    print("\nGiven array: ",heapsort_arr)
    class BinHeap:
        def __init__(self):
            self.heapList = [0]
            self.currentSize = 0
```

```
def HeapUp(self,i):
    while i // 2 > 0:
        if self.heapList[i] < self.heapList[i // 2]:
            tmp = self.heapList[i // 2]
            self.heapList[i // 2] = self.heapList[i]
            self.heapList[i] = tmp
        i = i // 2

def insert(self,k):
    self.heapList.append(k)
    self.currentSize = self.currentSize + 1
    self.HeapUp(self.currentSize)

def HeapDown(self,i):
    while (i * 2) <= self.currentSize:
        mc = self.minChild(i)
        if self.heapList[i] > self.heapList[mc]:
            tmp = self.heapList[i]
            self.heapList[i] = self.heapList[mc]
            self.heapList[mc] = tmp
        i = mc

def minChild(self,i):
    if i * 2 + 1 > self.currentSize:
        return i * 2
    else:
```

```
            if self.heapList[i*2] < self.heapList[i*2+1]:
                return i * 2
            else:
                return i * 2 + 1

    def delMin(self):
        retval = self.heapList[1]
        self.heapList[1] = self.heapList[self.currentSize]
        self.currentSize = self.currentSize - 1
        self.heapList.pop()
        self.HeapDown(1)
        return retval

    def buildHeap(self,alist):
        i = len(alist) // 2
        self.currentSize = len(alist)
        self.heapList = [0] + alist[:]
        while (i > 0):
            self.HeapDown(i)
            i = i - 1

def main():

    bh = BinHeap()
    bh.buildHeap(heapsort_arr)

    for i in range(len(heapsort_arr)):
```

```
        print(bh.delMin())

    main()
    end = None
    end=timeit.default_timer()
    Execution_time=end - start
    Execution_time_arr7.append(Execution_time)
    print("\nExecution Time in seconds:",Execution_time)

print("\nExecution Time for each run in seconds:\t",Execution_time_arr7)

#Plotting the graph for Heap sort with already sorted input array
plt.plot(val,Execution_time_arr7,label='Heap sort (Sorted Input)')
plt.xlabel('Input values')
plt.ylabel('Execution time(in seconds)')
plt.legend()
plt.show()

#If the array for heap sort is reversely sorted
Execution_time_arr8 = []
for j in val:
    heapsort_arr = []
    random.seed(3)
    heapsort_arr = random.sample(range(0,50000),j)
    start = None
    start = timeit.default_timer()
    #Sorting the array in reverse order
```

```
heapsort_arr = sorted(heapsort_arr,reverse=True)
print("\nGiven array: ",heapsort_arr)
class BinHeap:
    def __init__(self):
        self.heapList = [0]
        self.currentSize = 0

    def HeapUp(self,i):
        while i // 2 > 0:
            if self.heapList[i] < self.heapList[i // 2]:
                tmp = self.heapList[i // 2]
                self.heapList[i // 2] = self.heapList[i]
                self.heapList[i] = tmp
            i = i // 2

    def insert(self,k):
        self.heapList.append(k)
        self.currentSize = self.currentSize + 1
        self.HeapUp(self.currentSize)



    def HeapDown(self,i):
        while (i * 2) <= self.currentSize:
            mc = self.minChild(i)
            if self.heapList[i] > self.heapList[mc]:
                tmp = self.heapList[i]
                self.heapList[i] = self.heapList[mc]
```

```
            self.heapList[mc] = tmp
        i = mc

def minChild(self,i):
    if i * 2 + 1 > self.currentSize:
        return i * 2
    else:
        if self.heapList[i*2] < self.heapList[i*2+1]:
            return i * 2
        else:
            return i * 2 + 1



def delMin(self):
    retval = self.heapList[1]
    self.heapList[1] = self.heapList[self.currentSize]
    self.currentSize = self.currentSize - 1
    self.heapList.pop()
    self.HeapDown(1)
    return retval

def buildHeap(self,alist):
    i = len(alist) // 2
    self.currentSize = len(alist)
    self.heapList = [0] + alist[:]
    while (i > 0):
        self.HeapDown(i)
```

```
            i = i - 1

    def main():

        bh = BinHeap()
        bh.buildHeap(heapsort_arr)

        for i in range(len(heapsort_arr)):
            print(bh.delMin())

    main()
    end = None
    end=timeit.default_timer()
    Execution_time=end - start
    Execution_time_arr8.append(Execution_time)
    print("\nExecution Time in seconds:",Execution_time)

print("\nExecution Time for each run in seconds:\t",Execution_time_arr8)

#Plotting the graph for Heap sort with reversely sorted input array
plt.plot(val,Execution_time_arr8,label='Heap sort (Reverse sorted Input)')
plt.xlabel('Input values')
plt.ylabel('Execution time(in seconds)')
plt.legend()
plt.show()

#Comparison plot for Heap Sort
```

```
plt.plot(val,Execution_time_arr6,label='Heap sort (Random Input)')
plt.plot(val,Execution_time_arr7,label='Heap sort (Sorted Input)')
plt.plot(val,Execution_time_arr8,label='Heap sort (Reverse Sorted Input)')
plt.xlabel('Input values')
plt.ylabel('Execution time(in seconds)')
plt.legend()
plt.show()

#In-place Quick sort
Execution_time_arr9 = []
Execution_time_arr10 = []
Execution_time_arr11 = []
for j in val:
   quicksort_arr = []
   random.seed(3)
   quicksort_arr = random.sample(range(0,50000),j)
   start = None
   start = timeit.default_timer()
   print("\nGiven array: ",quicksort_arr)
   def quicksort_inplace(s, a, b):
      def swap(s, a, b):
         tmp = s[a]
         s[a] = s[b]
         s[b] = tmp
      if a >= b: return
      pivot = s[b] # last element as pivot
      left = a
```

```
    right = b - 1
    while left <= right:
        while left <= right and s[left] <= pivot:  # finding element larger than the pivot
            left += 1
        while left <= right and s[right] >= pivot: # finding element larger than the pivot
            right -= 1
        if left < right:
            swap(s, left, right)
    swap(s, left, b) # putting the pivot its final place
    quicksort_inplace(s, a, left-1)
    quicksort_inplace(s, left+1, b)

quicksort_inplace(quicksort_arr, 0, len(quicksort_arr)-1)
print ("\nSorted array", quicksort_arr)
end=None
end = timeit.default_timer()
Execution_time=end - start
Execution_time_arr9.append(Execution_time)
print("\nExecution Time in seconds:",Execution_time)

start=None
start = timeit.default_timer()
print("\nGiven array (Sorted input)",quicksort_arr)
def sorted_quicksort_inplace(s, a, b):
    quicksort_inplace(quicksort_arr, 0, len(quicksort_arr)-1)
    print ("\nSorted array(Sorted input)", quicksort_arr)
end=None
```

```
    end = timeit.default_timer()
    Execution_time=end - start
    Execution_time_arr10.append(Execution_time)
    print("\nExecution Time in seconds(Sorted Input):",Execution_time)

    start=None
    start = timeit.default_timer()
    quicksort_arr.reverse()
    print("\nGiven array (Reverse Sorted Input)",quicksort_arr)
    def reverse_sorted_quicksort_inplace(s, a, b):
        quicksort_inplace(quicksort_arr, 0, len(quicksort_arr)-1)
        print ("\nSorted array (Reverse Sorted Input)", quicksort_arr)
    end=None
    end = timeit.default_timer()
    Execution_time=end - start
    Execution_time_arr11.append(Execution_time)
    print("\nExecution Time in seconds(Reverse Sorted Input):",Execution_time)

print("\nExecution Time for each run in seconds:\t",Execution_time_arr9)
print("\nExecution Time for each run in seconds(Sorted):\t",Execution_time_arr10)
print("\nExecution Time for each run in seconds(Reverse Sorted):\t",Execution_time_arr11)

#Plotting the graph for In-place quicksort with random input array
plt.plot(val,Execution_time_arr9,label='Inplace Quick sort (Random Input)')
plt.xlabel('Input values')
plt.ylabel('Execution time(in seconds)')
plt.legend()
```

```
plt.show()

#Plotting the graph for In-place quicksort with already sorted input array
plt.plot(val,Execution_time_arr10,label='Inplace Quick sort (Sorted Input)')
plt.xlabel('Input values')
plt.ylabel('Execution time(in seconds)')
plt.legend()
plt.show()

#Plotting the graph for In-place quicksort with reversely sorted input array
plt.plot(val,Execution_time_arr11,label='Inplace Quick sort (Reversely sorted Input)')
plt.xlabel('Input values')
plt.ylabel('Execution time(in seconds)')
plt.legend()
plt.show()

#Comparison plot for In-place Quick Sort
plt.plot(val,Execution_time_arr9,label='In-place Quick sort (Random Input)')
plt.plot(val,Execution_time_arr10,label='In-place Quick sort (Sorted Input)')
plt.plot(val,Execution_time_arr11,label='In-place Quick sort (Reverse Sorted Input)')
plt.xlabel('Input values')
plt.ylabel('Execution time(in seconds)')
plt.legend()
plt.show()

#Modified Quick Sort
#A method to calculate the median of three numbers using two comparisons
```

```
Execution_time_arr12 = []
for j in val:
    quicksort_marr = []
    random.seed(3)
    quicksort_marr = random.sample(range(0,50000),j)
    start = None
    start = timeit.default_timer()
    print("\nGiven array: ",quicksort_marr)

    #finding median from the 3 numbers
    def median(a, b, c):
        if ( a - b) * (c - a) >= 0:
            return a

        elif (b - a) * (c - b) >= 0:
            return b

        else:
            return c

    #A method to partition around the median
    def partition_m(array, leftend, rightend):
        left = array[leftend]
        right = array[rightend-1]
        length = rightend - leftend
        if length % 2 == 0:
            middle = array[leftend + int(length/2) - 1]
```

```
    else:
        middle = array[leftend + int(length/2)]

    pivot = median(left, right, middle)

    pivotindex = array.index(pivot)

    array[pivotindex] = array[leftend]
    array[leftend] = pivot

    i = leftend + 1
    for j in range(leftend + 1, rightend):
        if array[j] < pivot:
            temp = array[j]
            array[j] = array[i]
            array[i] = temp
            i += 1

    leftendval = array[leftend]
    array[leftend] = array[i-1]
    array[i-1] = leftendval
    return i - 1

def quicksort_m(array, leftindex, rightindex):
    #For small sub-problem, calling insertion sort
    if(len(array)<=15):
        insertion_sort(quicksort_marr)
```

```
        elif(len(array)>15):
            if leftindex < rightindex:
                newpivotindex = partition_m(array, leftindex, rightindex)
                quicksort_m(array, leftindex, newpivotindex)
                quicksort_m(array, newpivotindex + 1, rightindex)

    quicksort_m(quicksort_marr, 0, len(quicksort_marr))
    print("\nQuick sort",quicksort_marr)
    end=None
    end = timeit.default_timer()
    Execution_time=end - start
    Execution_time_arr12.append(Execution_time)
    print("\nExecution Time in seconds:",Execution_time)
print("\nExecution Time for each run in seconds:\t",Execution_time_arr12)

#Plotting the graph for Modified quicksort with random input array
plt.plot(val,Execution_time_arr12,label='Modified Quick sort (Random Input)')
plt.xlabel('Input values')
plt.ylabel('Execution time(in seconds)')
plt.legend()
plt.show()

#If the array is already sorted
#Modified Quick Sort
#Calculating the median of three numbers using two comparisons
Execution_time_arr13 = []
for j in val:
```

```
quicksort_marr = []
random.seed(3)
quicksort_marr = random.sample(range(0,50000),j)
start = None
start = timeit.default_timer()
quicksort_marr=sorted(quicksort_marr)
print("\nGiven array: ",quicksort_marr)

def median(a, b, c):
    if ( a - b) * (c - a) >= 0:
        return a

    elif (b - a) * (c - b) >= 0:
        return b

    else:
        return c

#A method to partition around the median
def partition_median(array, leftend, rightend):
    left = array[leftend]
    right = array[rightend-1]
    length = rightend - leftend
    if length % 2 == 0:
        middle = array[leftend + int(length/2) - 1]
    else:
        middle = array[leftend + int(length/2)]
```

```
    pivot = median(left, right, middle)

    pivotindex = array.index(pivot)

    array[pivotindex] = array[leftend]
    array[leftend] = pivot

    i = leftend + 1
    for j in range(leftend + 1, rightend):
        if array[j] < pivot:
            temp = array[j]
            array[j] = array[i]
            array[i] = temp
            i += 1

    leftendval = array[leftend]
    array[leftend] = array[i-1]
    array[i-1] = leftendval
    return i - 1

def quicksort_median(array, leftindex, rightindex):
    #For small sub-problem, calling insertion sort
    if(len(array)<=15):
        insertion_sort(quicksort_marr)
    elif(len(array)>15):
         if leftindex < rightindex:
```

```
            newpivotindex = partition_median(array, leftindex, rightindex)
            quicksort_median(array, leftindex, newpivotindex)
            quicksort_median(array, newpivotindex + 1, rightindex)

    quicksort_median(quicksort_marr, 0, len(quicksort_marr))
    print("\nQuick sort",quicksort_marr)
    end=None
    end = timeit.default_timer()
    Execution_time=end - start
    Execution_time_arr13.append(Execution_time)
    print("\nExecution Time in seconds:",Execution_time)
print("\nExecution Time for each run in seconds:\t",Execution_time_arr13)

#Plotting the graph for Modified quicksort with already sorted input array
plt.plot(val,Execution_time_arr13,label='Modified Quick sort (Sorted Input)')
plt.xlabel('Input values')
plt.ylabel('Execution time(in seconds)')
plt.legend()
plt.show()

#If the array is reversely sorted
#Modified Quick Sort
Execution_time_arr14 = []
for j in val:
    quicksort_marr = []
    random.seed(3)
    quicksort_marr = random.sample(range(0,50000),j)
```

```
start = None
start = timeit.default_timer()
quicksort_marr.reverse()
print("\nGiven array: ",quicksort_marr)

def median(a, b, c):
    if ( a - b) * (c - a) >= 0:
        return a

    elif (b - a) * (c - b) >= 0:
        return b

    else:
        return c

#A method to partition around the median
def partition_median(array, leftend, rightend):
    left = array[leftend]
    right = array[rightend-1]
    length = rightend - leftend
    if length % 2 == 0:
        middle = array[leftend + int(length/2) - 1]
    else:
        middle = array[leftend + int(length/2)]

    pivot = median(left, right, middle)
```

```
    pivotindex = array.index(pivot)

    array[pivotindex] = array[leftend]
    array[leftend] = pivot

    i = leftend + 1
    for j in range(leftend + 1, rightend):
        if array[j] < pivot:
            temp = array[j]
            array[j] = array[i]
            array[i] = temp
            i += 1

    leftendval = array[leftend]
    array[leftend] = array[i-1]
    array[i-1] = leftendval
    return i - 1

def quicksort_median(array, leftindex, rightindex):
    if(len(array)<=15):
        insertion_sort(quicksort_marr)
    elif(len(array)>15):
        if leftindex < rightindex:
            newpivotindex = partition_median(array, leftindex, rightindex)
            quicksort_median(array, leftindex, newpivotindex)
            quicksort_median(array, newpivotindex + 1, rightindex)
```

```
    quicksort_median(quicksort_marr, 0, len(quicksort_marr))
    print("\nQuick sort",quicksort_marr)
    end=None
    end = timeit.default_timer()
    Execution_time=end - start
    Execution_time_arr14.append(Execution_time)
    print("\nExecution Time in seconds:",Execution_time)
print("\nExecution Time in seconds:\t",Execution_time_arr14)

#Plotting the graph for Modified quicksort with reversely sorted input array
plt.plot(val,Execution_time_arr14,label='Modified Quick sort (Reverse Sorted Input)')
plt.xlabel('Input values')
plt.ylabel('Execution time(in seconds)')
plt.legend()
plt.show()

#Comparison plot for Modified Quick Sort
plt.plot(val,Execution_time_arr12,label='Modified Quick sort (Random Input)')
plt.plot(val,Execution_time_arr13,label='Modified Quick sort (Sorted Input)')
plt.plot(val,Execution_time_arr14,label='Modified Quick sort (Reverse Sorted Input)')
plt.xlabel('Input values')
plt.ylabel('Execution time(in seconds)')
plt.legend()
plt.show()

#Comparison of all sorting algorithms for Random Input
plt.plot(val,Execution_time_arr,label='Insertion sort')
```

```
plt.plot(val,Execution_time_arr3,label='Merge Sort')
plt.plot(val,Execution_time_arr6,label='Heap Sort')
plt.plot(val,Execution_time_arr9,label='In-place Quick Sort')
plt.plot(val,Execution_time_arr12,label='Modified Quick sort')
plt.title('Comparison of all sorting algorithms for Random Input Values')
plt.xlabel('Input values')
plt.ylabel('Execution time(in seconds)')
plt.legend()
plt.show()

#Comparison of all sorting algorithms for Sorted Input
plt.plot(val,Execution_time_arr1,label='Insertion sort')
plt.plot(val,Execution_time_arr4,label='Merge Sort')
plt.plot(val,Execution_time_arr7,label='Heap Sort')
plt.plot(val,Execution_time_arr10,label='In-place Quick Sort')
plt.plot(val,Execution_time_arr13,label='Modified Quick sort')
plt.title('Comparison of all sorting algorithms for Sorted Input Values')
plt.xlabel('Input values')
plt.ylabel('Execution time(in seconds)')
plt.legend()
plt.show()

#Comparison of all sorting algorithms for Reversely Sorted Input
plt.plot(val,Execution_time_arr2,label='Insertion sort')
plt.plot(val,Execution_time_arr5,label='Merge Sort')
plt.plot(val,Execution_time_arr8,label='Heap Sort')
plt.plot(val,Execution_time_arr11,label='In-place Quick Sort')
```

plt.plot(val,Execution_time_arr14,label='Modified Quick sort')

plt.title('Comparison of all sorting algorithms for Reversely Input Values')

plt.xlabel('Input values')

plt.ylabel('Execution time(in seconds)')

plt.legend()

plt.show()

SCREENSHOTS OF EXECUTION TIMES FOR EACH ALGORITHM:

INSERTION SORT(RANDOM INPUT):

```
        print('\nThe sorted list after insertion sort: \t', s)
        print('\n')

    insertion_sort(insertion_sort_arr)
    end = None
    end = timeit.default_timer()

    Execution_time = end - start
    Execution_time_arr.append(Execution_time)
    print("Execution Time in seconds:",Execution_time)

print("Execution Time of each run:",Execution_time_arr)
```

```
49745, 49746, 49747, 49748, 49749, 49750, 49751, 49752, 49753, 49754, 49755, 49756, 49757, 49758, 49759, 49760, 49761, 49
762, 49763, 49764, 49765, 49766, 49767, 49768, 49769, 49770, 49771, 49772, 49773, 49774, 49775, 49776, 49777, 49778, 4977
9, 49780, 49781, 49782, 49783, 49784, 49785, 49786, 49787, 49788, 49789, 49790, 49791, 49792, 49793, 49794, 49795, 49796,
49797, 49798, 49799, 49800, 49801, 49802, 49803, 49804, 49805, 49806, 49807, 49808, 49809, 49810, 49811, 49812, 49813, 49
814, 49815, 49816, 49817, 49818, 49819, 49820, 49821, 49822, 49823, 49824, 49825, 49826, 49827, 49828, 49829, 49830, 4983
1, 49832, 49833, 49834, 49835, 49836, 49837, 49838, 49839, 49840, 49841, 49842, 49843, 49844, 49845, 49846, 49847, 49848,
49849, 49850, 49851, 49852, 49853, 49854, 49855, 49856, 49857, 49858, 49859, 49860, 49861, 49862, 49863, 49864, 49865, 49
866, 49867, 49868, 49869, 49870, 49871, 49872, 49873, 49874, 49875, 49876, 49877, 49878, 49879, 49880, 49881, 49882, 4988
3, 49884, 49885, 49886, 49887, 49888, 49889, 49890, 49891, 49892, 49893, 49894, 49895, 49896, 49897, 49898, 49899, 49900,
49901, 49902, 49903, 49904, 49905, 49906, 49907, 49908, 49909, 49910, 49911, 49912, 49913, 49914, 49915, 49916, 49917, 49
918, 49919, 49920, 49921, 49922, 49923, 49924, 49925, 49926, 49927, 49928, 49929, 49930, 49931, 49932, 49933, 49934, 4993
5, 49936, 49937, 49938, 49939, 49940, 49941, 49942, 49943, 49944, 49945, 49946, 49947, 49948, 49949, 49950, 49951, 49952,
49953, 49954, 49955, 49956, 49957, 49958, 49959, 49960, 49961, 49962, 49963, 49964, 49965, 49966, 49967, 49968, 49969, 49
970, 49971, 49972, 49973, 49974, 49975, 49976, 49977, 49978, 49979, 49980, 49981, 49982, 49983, 49984, 49985, 49986, 4998
7, 49988, 49989, 49990, 49991, 49992, 49993, 49994, 49995, 49996, 49997, 49998, 49999]


Execution Time in seconds: 242.6137761999853
Execution Time of each run: [0.1086516999930609, 0.5868847999954596, 1.179348999983631, 2.60425549998763, 9.5160822999896
48, 38.17438770001172, 87.23797479999484, 156.0567441000021, 242.6137761999853]
```

```
            s[j + 1] = key

        print('\nThe sorted list after insertion sort: \t', s)
        print('\n')

    insertion_sort(insertion_sort_arr)
    end = None
    end = timeit.default_timer()

    Execution_time = end - start
    Execution_time_arr.append(Execution_time)
    print("Execution Time in seconds:",Execution_time)

print("Execution Time of each run:",Execution_time_arr)
```

```
49745, 49746, 49747, 49748, 49749, 49750, 49751, 49752, 49753, 49754, 49755, 49756, 49757, 49758, 49759, 49760, 49761, 49
762, 49763, 49764, 49765, 49766, 49767, 49768, 49769, 49770, 49771, 49772, 49773, 49774, 49775, 49776, 49777, 49778, 4977
9, 49780, 49781, 49782, 49783, 49784, 49785, 49786, 49787, 49788, 49789, 49790, 49791, 49792, 49793, 49794, 49795, 49796,
49797, 49798, 49799, 49800, 49801, 49802, 49803, 49804, 49805, 49806, 49807, 49808, 49809, 49810, 49811, 49812, 49813, 49
814, 49815, 49816, 49817, 49818, 49819, 49820, 49821, 49822, 49823, 49824, 49825, 49826, 49827, 49828, 49829, 49830, 4983
1, 49832, 49833, 49834, 49835, 49836, 49837, 49838, 49839, 49840, 49841, 49842, 49843, 49844, 49845, 49846, 49847, 49848,
49849, 49850, 49851, 49852, 49853, 49854, 49855, 49856, 49857, 49858, 49859, 49860, 49861, 49862, 49863, 49864, 49865, 49
866, 49867, 49868, 49869, 49870, 49871, 49872, 49873, 49874, 49875, 49876, 49877, 49878, 49879, 49880, 49881, 49882, 4988
3, 49884, 49885, 49886, 49887, 49888, 49889, 49890, 49891, 49892, 49893, 49894, 49895, 49896, 49897, 49898, 49899, 49900,
49901, 49902, 49903, 49904, 49905, 49906, 49907, 49908, 49909, 49910, 49911, 49912, 49913, 49914, 49915, 49916, 49917, 49
918, 49919, 49920, 49921, 49922, 49923, 49924, 49925, 49926, 49927, 49928, 49929, 49930, 49931, 49932, 49933, 49934, 4993
5, 49936, 49937, 49938, 49939, 49940, 49941, 49942, 49943, 49944, 49945, 49946, 49947, 49948, 49949, 49950, 49951, 49952,
49953, 49954, 49955, 49956, 49957, 49958, 49959, 49960, 49961, 49962, 49963, 49964, 49965, 49966, 49967, 49968, 49969, 49
970, 49971, 49972, 49973, 49974, 49975, 49976, 49977, 49978, 49979, 49980, 49981, 49982, 49983, 49984, 49985, 49986, 4998
7, 49988, 49989, 49990, 49991, 49992, 49993, 49994, 49995, 49996, 49997, 49998, 49999]


Execution Time in seconds: 242.20376679999754
Execution Time of each run: [0.10975839997990988, 0.43080010000267066, 0.9155372999957763, 2.527153899980476, 10.37614420
0010458, 44.729661700024735, 90.3341264000046, 159.32198380000773, 242.20376679999754]
```

pynb#

```
        print('\nThe sorted list after insertion sort: \t', s)
        print('\n')

    insertion_sort(insertion_sort_arr)
    end = None
    end = timeit.default_timer()

    Execution_time = end - start
    Execution_time_arr.append(Execution_time)
    print("Execution Time in seconds:",Execution_time)

print("Execution Time of each run:",Execution_time_arr)
```

```
49745, 49746, 49747, 49748, 49749, 49750, 49751, 49752, 49753, 49754, 49755, 49756, 49757, 49758, 49759, 49760, 49761, 49
762, 49763, 49764, 49765, 49766, 49767, 49768, 49769, 49770, 49771, 49772, 49773, 49774, 49775, 49776, 49777, 49778, 4977
9, 49780, 49781, 49782, 49783, 49784, 49785, 49786, 49787, 49788, 49789, 49790, 49791, 49792, 49793, 49794, 49795, 49796,
49797, 49798, 49799, 49800, 49801, 49802, 49803, 49804, 49805, 49806, 49807, 49808, 49809, 49810, 49811, 49812, 49813, 49
814, 49815, 49816, 49817, 49818, 49819, 49820, 49821, 49822, 49823, 49824, 49825, 49826, 49827, 49828, 49829, 49830, 4983
1, 49832, 49833, 49834, 49835, 49836, 49837, 49838, 49839, 49840, 49841, 49842, 49843, 49844, 49845, 49846, 49847, 49848,
49849, 49850, 49851, 49852, 49853, 49854, 49855, 49856, 49857, 49858, 49859, 49860, 49861, 49862, 49863, 49864, 49865, 49
866, 49867, 49868, 49869, 49870, 49871, 49872, 49873, 49874, 49875, 49876, 49877, 49878, 49879, 49880, 49881, 49882, 4988
3, 49884, 49885, 49886, 49887, 49888, 49889, 49890, 49891, 49892, 49893, 49894, 49895, 49896, 49897, 49898, 49899, 49900,
49901, 49902, 49903, 49904, 49905, 49906, 49907, 49908, 49909, 49910, 49911, 49912, 49913, 49914, 49915, 49916, 49917, 49
918, 49919, 49920, 49921, 49922, 49923, 49924, 49925, 49926, 49927, 49928, 49929, 49930, 49931, 49932, 49933, 49934, 4993
5, 49936, 49937, 49938, 49939, 49940, 49941, 49942, 49943, 49944, 49945, 49946, 49947, 49948, 49949, 49950, 49951, 49952,
49953, 49954, 49955, 49956, 49957, 49958, 49959, 49960, 49961, 49962, 49963, 49964, 49965, 49966, 49967, 49968, 49969, 49
970, 49971, 49972, 49973, 49974, 49975, 49976, 49977, 49978, 49979, 49980, 49981, 49982, 49983, 49984, 49985, 49986, 4998
7, 49988, 49989, 49990, 49991, 49992, 49993, 49994, 49995, 49996, 49997, 49998, 49999]


Execution Time in seconds: 254.37130130000878
Execution Time of each run: [0.11868100002175197, 0.4458298999816179, 0.8935597999952734, 2.460764500021469, 9.7657973000
02305, 39.93786399997771, 94.80567249999149, 162.32418310001958, 254.37130130000878]
```

INSERTION SORT(SORTED INPUT):

```
                j -= 1
            s[j + 1] = key

        print('\nThe sorted list after insertion sort (sorted input array): \t', s)
        print('\n')

    insertion_sort(insertion_sort_arr)
    end = None
    end = timeit.default_timer()

    Execution_time = end - start
    Execution_time_arr1.append(Execution_time)
    print("Execution Time in seconds:",Execution_time)

print("Execution Time for each run in seconds:",Execution_time_arr1)
```

```
759, 49760, 49761, 49762, 49763, 49764, 49765, 49766, 49767, 49768, 49769, 49770, 49771, 49772, 49773, 49774, 49775, 4977
6, 49777, 49778, 49779, 49780, 49781, 49782, 49783, 49784, 49785, 49786, 49787, 49788, 49789, 49790, 49791, 49792, 49793,
49794, 49795, 49796, 49797, 49798, 49799, 49800, 49801, 49802, 49803, 49804, 49805, 49806, 49807, 49808, 49809, 49810, 49
811, 49812, 49813, 49814, 49815, 49816, 49817, 49818, 49819, 49820, 49821, 49822, 49823, 49824, 49825, 49826, 49827, 4982
8, 49829, 49830, 49831, 49832, 49833, 49834, 49835, 49836, 49837, 49838, 49839, 49840, 49841, 49842, 49843, 49844, 49845,
49846, 49847, 49848, 49849, 49850, 49851, 49852, 49853, 49854, 49855, 49856, 49857, 49858, 49859, 49860, 49861, 49862, 49
863, 49864, 49865, 49866, 49867, 49868, 49869, 49870, 49871, 49872, 49873, 49874, 49875, 49876, 49877, 49878, 49879, 4988
0, 49881, 49882, 49883, 49884, 49885, 49886, 49887, 49888, 49889, 49890, 49891, 49892, 49893, 49894, 49895, 49896, 49897,
49898, 49899, 49900, 49901, 49902, 49903, 49904, 49905, 49906, 49907, 49908, 49909, 49910, 49911, 49912, 49913, 49914, 49
915, 49916, 49917, 49918, 49919, 49920, 49921, 49922, 49923, 49924, 49925, 49926, 49927, 49928, 49929, 49930, 49931, 4993
2, 49933, 49934, 49935, 49936, 49937, 49938, 49939, 49940, 49941, 49942, 49943, 49944, 49945, 49946, 49947, 49948, 49949,
49950, 49951, 49952, 49953, 49954, 49955, 49956, 49957, 49958, 49959, 49960, 49961, 49962, 49963, 49964, 49965, 49966, 49
967, 49968, 49969, 49970, 49971, 49972, 49973, 49974, 49975, 49976, 49977, 49978, 49979, 49980, 49981, 49982, 49983, 4998
4, 49985, 49986, 49987, 49988, 49989, 49990, 49991, 49992, 49993, 49994, 49995, 49996, 49997, 49998, 49999]


Execution Time in seconds: 0.07112000000779517
Execution Time for each run in seconds: [0.0016681999841239303, 0.00437080001574941, 0.008061100001214072, 0.008279900008
346885, 0.016917400003876537, 0.027746700012357906, 0.056331300002057105, 0.04712410000502132, 0.07112000000779517]
```

```
        print('\nThe sorted list after insertion sort (sorted input array): \t', s)
        print('\n')

    insertion_sort(insertion_sort_arr)
    end = None
    end = timeit.default_timer()

    Execution_time = end - start
    Execution_time_arr1.append(Execution_time)
    print("Execution Time in seconds:",Execution_time)

print("Execution Time for each run in seconds:",Execution_time_arr1)
```

```
759, 49760, 49761, 49762, 49763, 49764, 49765, 49766, 49767, 49768, 49769, 49770, 49771, 49772, 49773, 49774, 49775, 4977
6, 49777, 49778, 49779, 49780, 49781, 49782, 49783, 49784, 49785, 49786, 49787, 49788, 49789, 49790, 49791, 49792, 49793,
49794, 49795, 49796, 49797, 49798, 49799, 49800, 49801, 49802, 49803, 49804, 49805, 49806, 49807, 49808, 49809, 49810, 49
811, 49812, 49813, 49814, 49815, 49816, 49817, 49818, 49819, 49820, 49821, 49822, 49823, 49824, 49825, 49826, 49827, 4982
8, 49829, 49830, 49831, 49832, 49833, 49834, 49835, 49836, 49837, 49838, 49839, 49840, 49841, 49842, 49843, 49844, 49845,
49846, 49847, 49848, 49849, 49850, 49851, 49852, 49853, 49854, 49855, 49856, 49857, 49858, 49859, 49860, 49861, 49862, 49
863, 49864, 49865, 49866, 49867, 49868, 49869, 49870, 49871, 49872, 49873, 49874, 49875, 49876, 49877, 49878, 49879, 4988
0, 49881, 49882, 49883, 49884, 49885, 49886, 49887, 49888, 49889, 49890, 49891, 49892, 49893, 49894, 49895, 49896, 49897,
49898, 49899, 49900, 49901, 49902, 49903, 49904, 49905, 49906, 49907, 49908, 49909, 49910, 49911, 49912, 49913, 49914, 49
915, 49916, 49917, 49918, 49919, 49920, 49921, 49922, 49923, 49924, 49925, 49926, 49927, 49928, 49929, 49930, 49931, 4993
2, 49933, 49934, 49935, 49936, 49937, 49938, 49939, 49940, 49941, 49942, 49943, 49944, 49945, 49946, 49947, 49948, 49949,
49950, 49951, 49952, 49953, 49954, 49955, 49956, 49957, 49958, 49959, 49960, 49961, 49962, 49963, 49964, 49965, 49966, 49
967, 49968, 49969, 49970, 49971, 49972, 49973, 49974, 49975, 49976, 49977, 49978, 49979, 49980, 49981, 49982, 49983, 4998
4, 49985, 49986, 49987, 49988, 49989, 49990, 49991, 49992, 49993, 49994, 49995, 49996, 49997, 49998, 49999]


Execution Time in seconds: 0.08038810000289232
Execution Time for each run in seconds: [0.002869200019631535, 0.002878100000890344, 0.00471889998880215, 0.006699699995
806441, 0.0164101000118535, 0.026552899973466992, 0.055049699993105605, 0.04876720000174828, 0.08038810000289232]
```

```
        print('\nThe sorted list after insertion sort (sorted input array): \t', s)
        print('\n')

    insertion_sort(insertion_sort_arr)
    end = None
    end = timeit.default_timer()

    Execution_time = end - start
    Execution_time_arr1.append(Execution_time)
    print("Execution Time in seconds:",Execution_time)

print("Execution Time for each run in seconds:",Execution_time_arr1)
```

```
6, 49777, 49778, 49779, 49780, 49781, 49782, 49783, 49784, 49785, 49786, 49787, 49788, 49789, 49790, 49791, 49792, 49793,
49794, 49795, 49796, 49797, 49798, 49799, 49800, 49801, 49802, 49803, 49804, 49805, 49806, 49807, 49808, 49809, 49810, 49
811, 49812, 49813, 49814, 49815, 49816, 49817, 49818, 49819, 49820, 49821, 49822, 49823, 49824, 49825, 49826, 49827, 4982
8, 49829, 49830, 49831, 49832, 49833, 49834, 49835, 49836, 49837, 49838, 49839, 49840, 49841, 49842, 49843, 49844, 49845,
49846, 49847, 49848, 49849, 49850, 49851, 49852, 49853, 49854, 49855, 49856, 49857, 49858, 49859, 49860, 49861, 49862, 49
863, 49864, 49865, 49866, 49867, 49868, 49869, 49870, 49871, 49872, 49873, 49874, 49875, 49876, 49877, 49878, 49879, 4988
0, 49881, 49882, 49883, 49884, 49885, 49886, 49887, 49888, 49889, 49890, 49891, 49892, 49893, 49894, 49895, 49896, 49897,
49898, 49899, 49900, 49901, 49902, 49903, 49904, 49905, 49906, 49907, 49908, 49909, 49910, 49911, 49912, 49913, 49914, 49
915, 49916, 49917, 49918, 49919, 49920, 49921, 49922, 49923, 49924, 49925, 49926, 49927, 49928, 49929, 49930, 49931, 4993
2, 49933, 49934, 49935, 49936, 49937, 49938, 49939, 49940, 49941, 49942, 49943, 49944, 49945, 49946, 49947, 49948, 49949,
49950, 49951, 49952, 49953, 49954, 49955, 49956, 49957, 49958, 49959, 49960, 49961, 49962, 49963, 49964, 49965, 49966, 49
967, 49968, 49969, 49970, 49971, 49972, 49973, 49974, 49975, 49976, 49977, 49978, 49979, 49980, 49981, 49982, 49983, 4998
4, 49985, 49986, 49987, 49988, 49989, 49990, 49991, 49992, 49993, 49994, 49995, 49996, 49997, 49998, 49999]


Execution Time in seconds: 0.08727910000015981
Execution Time for each run in seconds: [0.0014130999916233122, 0.004405999992741272, 0.004959800018696114, 0.00923529997
8172407, 0.017860299994936213, 0.03255410000565462, 0.051456900022458285, 0.054769499984104186, 0.08727910000015981]
```

INSERTION SORT(REVERSE INPUT):

```
        for i in range(1, len(s)):
            key = s[i]
            j = i - 1
            while j >= 0 and key < s[j]:
                s[j + 1] = s[j]
                j -= 1
            s[j + 1] = key

        print('\nThe sorted list after insertion sort(Reversely Sorted Input): \t', s)
        print('\n')

    insertion_sort(insertion_sort_arr)
    end = None
    end = timeit.default_timer()
    Execution_time = end - start
    Execution_time_arr2.append(Execution_time)
    print("Execution Time in seconds:",Execution_time)

print("Execution Time in seconds:",Execution_time_arr2)
```

```
762, 49763, 49764, 49765, 49766, 49767, 49768, 49769, 49770, 49771, 49772, 49773, 49774, 49775, 49776, 49777, 49778, 4977
9, 49780, 49781, 49782, 49783, 49784, 49785, 49786, 49787, 49788, 49789, 49790, 49791, 49792, 49793, 49794, 49795, 49796,
49797, 49798, 49799, 49800, 49801, 49802, 49803, 49804, 49805, 49806, 49807, 49808, 49809, 49810, 49811, 49812, 49813, 49
814, 49815, 49816, 49817, 49818, 49819, 49820, 49821, 49822, 49823, 49824, 49825, 49826, 49827, 49828, 49829, 49830, 4983
1, 49832, 49833, 49834, 49835, 49836, 49837, 49838, 49839, 49840, 49841, 49842, 49843, 49844, 49845, 49846, 49847, 49848,
49849, 49850, 49851, 49852, 49853, 49854, 49855, 49856, 49857, 49858, 49859, 49860, 49861, 49862, 49863, 49864, 49865, 49
866, 49867, 49868, 49869, 49870, 49871, 49872, 49873, 49874, 49875, 49876, 49877, 49878, 49879, 49880, 49881, 49882, 4988
3, 49884, 49885, 49886, 49887, 49888, 49889, 49890, 49891, 49892, 49893, 49894, 49895, 49896, 49897, 49898, 49899, 49900,
49901, 49902, 49903, 49904, 49905, 49906, 49907, 49908, 49909, 49910, 49911, 49912, 49913, 49914, 49915, 49916, 49917, 49
918, 49919, 49920, 49921, 49922, 49923, 49924, 49925, 49926, 49927, 49928, 49929, 49930, 49931, 49932, 49933, 49934, 4993
5, 49936, 49937, 49938, 49939, 49940, 49941, 49942, 49943, 49944, 49945, 49946, 49947, 49948, 49949, 49950, 49951, 49952,
49953, 49954, 49955, 49956, 49957, 49958, 49959, 49960, 49961, 49962, 49963, 49964, 49965, 49966, 49967, 49968, 49969, 49
970, 49971, 49972, 49973, 49974, 49975, 49976, 49977, 49978, 49979, 49980, 49981, 49982, 49983, 49984, 49985, 49986, 4998
7, 49988, 49989, 49990, 49991, 49992, 49993, 49994, 49995, 49996, 49997, 49998, 49999]


Execution Time in seconds: 481.82338059999165
Execution Time in seconds: [0.19345089999842457, 0.7592545000079554, 1.736934900021879, 4.766207300010137, 19.42224800001
7134, 78.9479083000042, 176.35081329999957, 319.3950360000017, 481.82338059999165]
```

```
        print('\nThe sorted list after insertion sort(Reversely Sorted Input): \t', s)
        print('\n')

    insertion_sort(insertion_sort_arr)
    end = None
    end = timeit.default_timer()
    Execution_time = end - start
    Execution_time_arr2.append(Execution_time)
    print("Execution Time in seconds:",Execution_time)

print("Execution Time in seconds:",Execution_time_arr2)
```

```
0, 49741, 49742, 49743, 49744, 49745, 49746, 49747, 49748, 49749, 49750, 49751, 49752, 49753, 49754, 49755, 49756, 49757,
49758, 49759, 49760, 49761, 49762, 49763, 49764, 49765, 49766, 49767, 49768, 49769, 49770, 49771, 49772, 49773, 49774, 49
775, 49776, 49777, 49778, 49779, 49780, 49781, 49782, 49783, 49784, 49785, 49786, 49787, 49788, 49789, 49790, 49791, 4979
2, 49793, 49794, 49795, 49796, 49797, 49798, 49799, 49800, 49801, 49802, 49803, 49804, 49805, 49806, 49807, 49808, 49809,
49810, 49811, 49812, 49813, 49814, 49815, 49816, 49817, 49818, 49819, 49820, 49821, 49822, 49823, 49824, 49825, 49826, 49
827, 49828, 49829, 49830, 49831, 49832, 49833, 49834, 49835, 49836, 49837, 49838, 49839, 49840, 49841, 49842, 49843, 4984
4, 49845, 49846, 49847, 49848, 49849, 49850, 49851, 49852, 49853, 49854, 49855, 49856, 49857, 49858, 49859, 49860, 49861,
49862, 49863, 49864, 49865, 49866, 49867, 49868, 49869, 49870, 49871, 49872, 49873, 49874, 49875, 49876, 49877, 49878, 49
879, 49880, 49881, 49882, 49883, 49884, 49885, 49886, 49887, 49888, 49889, 49890, 49891, 49892, 49893, 49894, 49895, 4989
6, 49897, 49898, 49899, 49900, 49901, 49902, 49903, 49904, 49905, 49906, 49907, 49908, 49909, 49910, 49911, 49912, 49913,
49914, 49915, 49916, 49917, 49918, 49919, 49920, 49921, 49922, 49923, 49924, 49925, 49926, 49927, 49928, 49929, 49930, 49
931, 49932, 49933, 49934, 49935, 49936, 49937, 49938, 49939, 49940, 49941, 49942, 49943, 49944, 49945, 49946, 49947, 4994
8, 49949, 49950, 49951, 49952, 49953, 49954, 49955, 49956, 49957, 49958, 49959, 49960, 49961, 49962, 49963, 49964, 49965,
49966, 49967, 49968, 49969, 49970, 49971, 49972, 49973, 49974, 49975, 49976, 49977, 49978, 49979, 49980, 49981, 49982, 49
983, 49984, 49985, 49986, 49987, 49988, 49989, 49990, 49991, 49992, 49993, 49994, 49995, 49996, 49997, 49998, 49999]


Execution Time in seconds: 492.29886159999296
Execution Time in seconds: [0.18809970002621412, 0.7656358999956865, 1.7145240000099875, 4.964721299998928, 20.1031487000
0002, 78.92833009999595, 182.65913749998435, 314.7004966000095, 492.29886159999296]
```

```
            s[j + 1] = key

        print('\nThe sorted list after insertion sort(Reversely Sorted Input): \t', s)
        print('\n')

    insertion_sort(insertion_sort_arr)
    end = None
    end = timeit.default_timer()
    Execution_time = end - start
    Execution_time_arr2.append(Execution_time)
    print("Execution Time in seconds:",Execution_time)

print("Execution Time in seconds:",Execution_time_arr2)
```

```
0, 49741, 49742, 49743, 49744, 49745, 49746, 49747, 49748, 49749, 49750, 49751, 49752, 49753, 49754, 49755, 49756, 49757,
49758, 49759, 49760, 49761, 49762, 49763, 49764, 49765, 49766, 49767, 49768, 49769, 49770, 49771, 49772, 49773, 49774, 49
775, 49776, 49777, 49778, 49779, 49780, 49781, 49782, 49783, 49784, 49785, 49786, 49787, 49788, 49789, 49790, 49791, 4979
2, 49793, 49794, 49795, 49796, 49797, 49798, 49799, 49800, 49801, 49802, 49803, 49804, 49805, 49806, 49807, 49808, 49809,
49810, 49811, 49812, 49813, 49814, 49815, 49816, 49817, 49818, 49819, 49820, 49821, 49822, 49823, 49824, 49825, 49826, 49
827, 49828, 49829, 49830, 49831, 49832, 49833, 49834, 49835, 49836, 49837, 49838, 49839, 49840, 49841, 49842, 49843, 4984
4, 49845, 49846, 49847, 49848, 49849, 49850, 49851, 49852, 49853, 49854, 49855, 49856, 49857, 49858, 49859, 49860, 49861,
49862, 49863, 49864, 49865, 49866, 49867, 49868, 49869, 49870, 49871, 49872, 49873, 49874, 49875, 49876, 49877, 49878, 49
879, 49880, 49881, 49882, 49883, 49884, 49885, 49886, 49887, 49888, 49889, 49890, 49891, 49892, 49893, 49894, 49895, 4989
6, 49897, 49898, 49899, 49900, 49901, 49902, 49903, 49904, 49905, 49906, 49907, 49908, 49909, 49910, 49911, 49912, 49913,
49914, 49915, 49916, 49917, 49918, 49919, 49920, 49921, 49922, 49923, 49924, 49925, 49926, 49927, 49928, 49929, 49930, 49
931, 49932, 49933, 49934, 49935, 49936, 49937, 49938, 49939, 49940, 49941, 49942, 49943, 49944, 49945, 49946, 49947, 4994
8, 49949, 49950, 49951, 49952, 49953, 49954, 49955, 49956, 49957, 49958, 49959, 49960, 49961, 49962, 49963, 49964, 49965,
49966, 49967, 49968, 49969, 49970, 49971, 49972, 49973, 49974, 49975, 49976, 49977, 49978, 49979, 49980, 49981, 49982, 49
983, 49984, 49985, 49986, 49987, 49988, 49989, 49990, 49991, 49992, 49993, 49994, 49995, 49996, 49997, 49998, 49999]


Execution Time in seconds: 482.35612040001433
Execution Time in seconds: [0.24215199999161996, 0.768554000009317, 1.6830944000103045, 4.9967670000041835, 20.5996700999
9673, 79.10216090001632, 177.3339486999903, 316.6799426000216, 482.35612040001433]
```

MERGE SORT(RANDOM INPUT):

```
    sorted_array = mergesort(merge_sort_arr)

    print(sorted_array)
    end = None
    end = timeit.default_timer()
    Execution_time=end - start
    Execution_time_arr3.append(Execution_time)
    print("\nExecution Time in seconds:\t",Execution_time)

print("\nExecution Time in seconds:\t",Execution_time_arr3)
```

```
7, 49768, 49769, 49770, 49771, 49772, 49773, 49774, 49775, 49776, 49777, 49778, 49779, 49780, 49781, 49782, 49783, 49784,
49785, 49786, 49787, 49788, 49789, 49790, 49791, 49792, 49793, 49794, 49795, 49796, 49797, 49798, 49799, 49800, 49801, 49
802, 49803, 49804, 49805, 49806, 49807, 49808, 49809, 49810, 49811, 49812, 49813, 49814, 49815, 49816, 49817, 49818, 4981
9, 49820, 49821, 49822, 49823, 49824, 49825, 49826, 49827, 49828, 49829, 49830, 49831, 49832, 49833, 49834, 49835, 49836,
49837, 49838, 49839, 49840, 49841, 49842, 49843, 49844, 49845, 49846, 49847, 49848, 49849, 49850, 49851, 49852, 49853, 49
854, 49855, 49856, 49857, 49858, 49859, 49860, 49861, 49862, 49863, 49864, 49865, 49866, 49867, 49868, 49869, 49870, 4987
1, 49872, 49873, 49874, 49875, 49876, 49877, 49878, 49879, 49880, 49881, 49882, 49883, 49884, 49885, 49886, 49887, 49888,
49889, 49890, 49891, 49892, 49893, 49894, 49895, 49896, 49897, 49898, 49899, 49900, 49901, 49902, 49903, 49904, 49905, 49
906, 49907, 49908, 49909, 49910, 49911, 49912, 49913, 49914, 49915, 49916, 49917, 49918, 49919, 49920, 49921, 49922, 4992
3, 49924, 49925, 49926, 49927, 49928, 49929, 49930, 49931, 49932, 49933, 49934, 49935, 49936, 49937, 49938, 49939, 49940,
49941, 49942, 49943, 49944, 49945, 49946, 49947, 49948, 49949, 49950, 49951, 49952, 49953, 49954, 49955, 49956, 49957, 49
958, 49959, 49960, 49961, 49962, 49963, 49964, 49965, 49966, 49967, 49968, 49969, 49970, 49971, 49972, 49973, 49974, 4997
5, 49976, 49977, 49978, 49979, 49980, 49981, 49982, 49983, 49984, 49985, 49986, 49987, 49988, 49989, 49990, 49991, 49992,
49993, 49994, 49995, 49996, 49997, 49998, 49999]

Execution Time in seconds:      0.3995983999921009

Execution Time in seconds:      [0.006606300012208521, 0.00976230000378564, 0.014904099982231855, 0.025774399982765317,
0.04750220000278205, 0.0996852999960538, 0.1685765000001993, 0.2591091000067536, 0.3995983999921009]
```

```
    sorted_array = mergesort(merge_sort_arr)

    print(sorted_array)
    end = None
    end = timeit.default_timer()
    Execution_time=end - start
    Execution_time_arr3.append(Execution_time)
    print("\nExecution Time in seconds:\t",Execution_time)

print("\nExecution Time in seconds:\t",Execution_time_arr3)
```

```
7, 49768, 49769, 49770, 49771, 49772, 49773, 49774, 49775, 49776, 49777, 49778, 49779, 49780, 49781, 49782, 49783, 49784,
49785, 49786, 49787, 49788, 49789, 49790, 49791, 49792, 49793, 49794, 49795, 49796, 49797, 49798, 49799, 49800, 49801, 49
802, 49803, 49804, 49805, 49806, 49807, 49808, 49809, 49810, 49811, 49812, 49813, 49814, 49815, 49816, 49817, 49818, 4981
9, 49820, 49821, 49822, 49823, 49824, 49825, 49826, 49827, 49828, 49829, 49830, 49831, 49832, 49833, 49834, 49835, 49836,
49837, 49838, 49839, 49840, 49841, 49842, 49843, 49844, 49845, 49846, 49847, 49848, 49849, 49850, 49851, 49852, 49853, 49
854, 49855, 49856, 49857, 49858, 49859, 49860, 49861, 49862, 49863, 49864, 49865, 49866, 49867, 49868, 49869, 49870, 4987
1, 49872, 49873, 49874, 49875, 49876, 49877, 49878, 49879, 49880, 49881, 49882, 49883, 49884, 49885, 49886, 49887, 49888,
49889, 49890, 49891, 49892, 49893, 49894, 49895, 49896, 49897, 49898, 49899, 49900, 49901, 49902, 49903, 49904, 49905, 49
906, 49907, 49908, 49909, 49910, 49911, 49912, 49913, 49914, 49915, 49916, 49917, 49918, 49919, 49920, 49921, 49922, 4992
3, 49924, 49925, 49926, 49927, 49928, 49929, 49930, 49931, 49932, 49933, 49934, 49935, 49936, 49937, 49938, 49939, 49940,
49941, 49942, 49943, 49944, 49945, 49946, 49947, 49948, 49949, 49950, 49951, 49952, 49953, 49954, 49955, 49956, 49957, 49
958, 49959, 49960, 49961, 49962, 49963, 49964, 49965, 49966, 49967, 49968, 49969, 49970, 49971, 49972, 49973, 49974, 4997
5, 49976, 49977, 49978, 49979, 49980, 49981, 49982, 49983, 49984, 49985, 49986, 49987, 49988, 49989, 49990, 49991, 49992,
49993, 49994, 49995, 49996, 49997, 49998, 49999]

Execution Time in seconds:      0.3002658999757841

Execution Time in seconds:      [0.005286200001137331, 0.00862440001219511, 0.014360800007125363, 0.026711300015449524,
0.04762220001430251, 0.10334649999276735, 0.1829557000019122, 0.24452569999266416, 0.3002658999757841]
```

```
    sorted_array = mergesort(merge_sort_arr)

    print(sorted_array)
    end = None
    end = timeit.default_timer()
    Execution_time=end - start
    Execution_time_arr3.append(Execution_time)
    print("\nExecution Time in seconds:\t",Execution_time)

print("\nExecution Time in seconds:\t",Execution_time_arr3)
```

```
7, 49768, 49769, 49770, 49771, 49772, 49773, 49774, 49775, 49776, 49777, 49778, 49779, 49780, 49781, 49782, 49783, 49784,
49785, 49786, 49787, 49788, 49789, 49790, 49791, 49792, 49793, 49794, 49795, 49796, 49797, 49798, 49799, 49800, 49801, 49
802, 49803, 49804, 49805, 49806, 49807, 49808, 49809, 49810, 49811, 49812, 49813, 49814, 49815, 49816, 49817, 49818, 4981
9, 49820, 49821, 49822, 49823, 49824, 49825, 49826, 49827, 49828, 49829, 49830, 49831, 49832, 49833, 49834, 49835, 49836,
49837, 49838, 49839, 49840, 49841, 49842, 49843, 49844, 49845, 49846, 49847, 49848, 49849, 49850, 49851, 49852, 49853, 49
854, 49855, 49856, 49857, 49858, 49859, 49860, 49861, 49862, 49863, 49864, 49865, 49866, 49867, 49868, 49869, 49870, 4987
1, 49872, 49873, 49874, 49875, 49876, 49877, 49878, 49879, 49880, 49881, 49882, 49883, 49884, 49885, 49886, 49887, 49888,
49889, 49890, 49891, 49892, 49893, 49894, 49895, 49896, 49897, 49898, 49899, 49900, 49901, 49902, 49903, 49904, 49905, 49
906, 49907, 49908, 49909, 49910, 49911, 49912, 49913, 49914, 49915, 49916, 49917, 49918, 49919, 49920, 49921, 49922, 4992
3, 49924, 49925, 49926, 49927, 49928, 49929, 49930, 49931, 49932, 49933, 49934, 49935, 49936, 49937, 49938, 49939, 49940,
49941, 49942, 49943, 49944, 49945, 49946, 49947, 49948, 49949, 49950, 49951, 49952, 49953, 49954, 49955, 49956, 49957, 49
958, 49959, 49960, 49961, 49962, 49963, 49964, 49965, 49966, 49967, 49968, 49969, 49970, 49971, 49972, 49973, 49974, 4997
5, 49976, 49977, 49978, 49979, 49980, 49981, 49982, 49983, 49984, 49985, 49986, 49987, 49988, 49989, 49990, 49991, 49992,
49993, 49994, 49995, 49996, 49997, 49998, 49999]

Execution Time in seconds:      0.3041339999763295

Execution Time in seconds:      [0.007652700005564839, 0.00940960002481006, 0.014113099983660504, 0.02626439998857677,
0.05051110000931658, 0.11411329999100417, 0.18504420001409017, 0.25002420000964776, 0.3041339999763295]
```

MERGE SORT(SORTED INPUT):

```
    sorted_array = mergesort(merge_sort_arr)

    print("\nSorted array:",sorted_array)
    end = None
    end = timeit.default_timer()
    Execution_time=end - start
    Execution_time_arr4.append(Execution_time)
    print("\nExecution Time in seconds:\t",Execution_time)

print("\nExecution Time in seconds:\t",Execution_time_arr4)
```

```
748, 49749, 49750, 49751, 49752, 49753, 49754, 49755, 49756, 49757, 49758, 49759, 49760, 49761, 49762, 49763, 49764, 4976
5, 49766, 49767, 49768, 49769, 49770, 49771, 49772, 49773, 49774, 49775, 49776, 49777, 49778, 49779, 49780, 49781, 49782,
49783, 49784, 49785, 49786, 49787, 49788, 49789, 49790, 49791, 49792, 49793, 49794, 49795, 49796, 49797, 49798, 49799, 49
800, 49801, 49802, 49803, 49804, 49805, 49806, 49807, 49808, 49809, 49810, 49811, 49812, 49813, 49814, 49815, 49816, 4981
7, 49818, 49819, 49820, 49821, 49822, 49823, 49824, 49825, 49826, 49827, 49828, 49829, 49830, 49831, 49832, 49833, 49834,
49835, 49836, 49837, 49838, 49839, 49840, 49841, 49842, 49843, 49844, 49845, 49846, 49847, 49848, 49849, 49850, 49851, 49
852, 49853, 49854, 49855, 49856, 49857, 49858, 49859, 49860, 49861, 49862, 49863, 49864, 49865, 49866, 49867, 49868, 4986
9, 49870, 49871, 49872, 49873, 49874, 49875, 49876, 49877, 49878, 49879, 49880, 49881, 49882, 49883, 49884, 49885, 49886,
49887, 49888, 49889, 49890, 49891, 49892, 49893, 49894, 49895, 49896, 49897, 49898, 49899, 49900, 49901, 49902, 49903, 49
904, 49905, 49906, 49907, 49908, 49909, 49910, 49911, 49912, 49913, 49914, 49915, 49916, 49917, 49918, 49919, 49920, 4992
1, 49922, 49923, 49924, 49925, 49926, 49927, 49928, 49929, 49930, 49931, 49932, 49933, 49934, 49935, 49936, 49937, 49938,
49939, 49940, 49941, 49942, 49943, 49944, 49945, 49946, 49947, 49948, 49949, 49950, 49951, 49952, 49953, 49954, 49955, 49
956, 49957, 49958, 49959, 49960, 49961, 49962, 49963, 49964, 49965, 49966, 49967, 49968, 49969, 49970, 49971, 49972, 4997
3, 49974, 49975, 49976, 49977, 49978, 49979, 49980, 49981, 49982, 49983, 49984, 49985, 49986, 49987, 49988, 49989, 49990,
49991, 49992, 49993, 49994, 49995, 49996, 49997, 49998, 49999]

Execution Time in seconds:      0.2882347999839112

Execution Time in seconds:      [0.005477200000314042, 0.009312699985457584, 0.014645300019765273, 0.024199799983762205,
0.04713859999901615, 0.10253350000130013, 0.175393300014548, 0.2439740999834612, 0.2882347999839112]
```

```
    sorted_array = mergesort(merge_sort_arr)

    print("\nSorted array:",sorted_array)
    end = None
    end = timeit.default_timer()
    Execution_time=end - start
    Execution_time_arr4.append(Execution_time)
    print("\nExecution Time in seconds:\t",Execution_time)

print("\nExecution Time in seconds:\t",Execution_time_arr4)
```

```
748, 49749, 49750, 49751, 49752, 49753, 49754, 49755, 49756, 49757, 49758, 49759, 49760, 49761, 49762, 49763, 49764, 4976
5, 49766, 49767, 49768, 49769, 49770, 49771, 49772, 49773, 49774, 49775, 49776, 49777, 49778, 49779, 49780, 49781, 49782,
49783, 49784, 49785, 49786, 49787, 49788, 49789, 49790, 49791, 49792, 49793, 49794, 49795, 49796, 49797, 49798, 49799, 49
800, 49801, 49802, 49803, 49804, 49805, 49806, 49807, 49808, 49809, 49810, 49811, 49812, 49813, 49814, 49815, 49816, 4981
7, 49818, 49819, 49820, 49821, 49822, 49823, 49824, 49825, 49826, 49827, 49828, 49829, 49830, 49831, 49832, 49833, 49834,
49835, 49836, 49837, 49838, 49839, 49840, 49841, 49842, 49843, 49844, 49845, 49846, 49847, 49848, 49849, 49850, 49851, 49
852, 49853, 49854, 49855, 49856, 49857, 49858, 49859, 49860, 49861, 49862, 49863, 49864, 49865, 49866, 49867, 49868, 4986
9, 49870, 49871, 49872, 49873, 49874, 49875, 49876, 49877, 49878, 49879, 49880, 49881, 49882, 49883, 49884, 49885, 49886,
49887, 49888, 49889, 49890, 49891, 49892, 49893, 49894, 49895, 49896, 49897, 49898, 49899, 49900, 49901, 49902, 49903, 49
904, 49905, 49906, 49907, 49908, 49909, 49910, 49911, 49912, 49913, 49914, 49915, 49916, 49917, 49918, 49919, 49920, 4992
1, 49922, 49923, 49924, 49925, 49926, 49927, 49928, 49929, 49930, 49931, 49932, 49933, 49934, 49935, 49936, 49937, 49938,
49939, 49940, 49941, 49942, 49943, 49944, 49945, 49946, 49947, 49948, 49949, 49950, 49951, 49952, 49953, 49954, 49955, 49
956, 49957, 49958, 49959, 49960, 49961, 49962, 49963, 49964, 49965, 49966, 49967, 49968, 49969, 49970, 49971, 49972, 4997
3, 49974, 49975, 49976, 49977, 49978, 49979, 49980, 49981, 49982, 49983, 49984, 49985, 49986, 49987, 49988, 49989, 49990,
49991, 49992, 49993, 49994, 49995, 49996, 49997, 49998, 49999]

Execution Time in seconds:       0.2910838999960106

Execution Time in seconds:       [0.005651000014040619, 0.009203499997965991, 0.017400400014594197, 0.02924840000923723,
0.05275880001136102, 0.10965030000079423, 0.19061409999267198, 0.24224839999806136, 0.2910838999960106]
```

```
    sorted_array = mergesort(merge_sort_arr)

    print("\nSorted array:",sorted_array)
    end = None
    end = timeit.default_timer()
    Execution_time=end - start
    Execution_time_arr4.append(Execution_time)
    print("\nExecution Time in seconds:\t",Execution_time)

print("\nExecution Time in seconds:\t",Execution_time_arr4)
```

```
748, 49749, 49750, 49751, 49752, 49753, 49754, 49755, 49756, 49757, 49758, 49759, 49760, 49761, 49762, 49763, 49764, 4976
5, 49766, 49767, 49768, 49769, 49770, 49771, 49772, 49773, 49774, 49775, 49776, 49777, 49778, 49779, 49780, 49781, 49782,
49783, 49784, 49785, 49786, 49787, 49788, 49789, 49790, 49791, 49792, 49793, 49794, 49795, 49796, 49797, 49798, 49799, 49
800, 49801, 49802, 49803, 49804, 49805, 49806, 49807, 49808, 49809, 49810, 49811, 49812, 49813, 49814, 49815, 49816, 4981
7, 49818, 49819, 49820, 49821, 49822, 49823, 49824, 49825, 49826, 49827, 49828, 49829, 49830, 49831, 49832, 49833, 49834,
49835, 49836, 49837, 49838, 49839, 49840, 49841, 49842, 49843, 49844, 49845, 49846, 49847, 49848, 49849, 49850, 49851, 49
852, 49853, 49854, 49855, 49856, 49857, 49858, 49859, 49860, 49861, 49862, 49863, 49864, 49865, 49866, 49867, 49868, 4986
9, 49870, 49871, 49872, 49873, 49874, 49875, 49876, 49877, 49878, 49879, 49880, 49881, 49882, 49883, 49884, 49885, 49886,
49887, 49888, 49889, 49890, 49891, 49892, 49893, 49894, 49895, 49896, 49897, 49898, 49899, 49900, 49901, 49902, 49903, 49
904, 49905, 49906, 49907, 49908, 49909, 49910, 49911, 49912, 49913, 49914, 49915, 49916, 49917, 49918, 49919, 49920, 4992
1, 49922, 49923, 49924, 49925, 49926, 49927, 49928, 49929, 49930, 49931, 49932, 49933, 49934, 49935, 49936, 49937, 49938,
49939, 49940, 49941, 49942, 49943, 49944, 49945, 49946, 49947, 49948, 49949, 49950, 49951, 49952, 49953, 49954, 49955, 49
956, 49957, 49958, 49959, 49960, 49961, 49962, 49963, 49964, 49965, 49966, 49967, 49968, 49969, 49970, 49971, 49972, 4997
3, 49974, 49975, 49976, 49977, 49978, 49979, 49980, 49981, 49982, 49983, 49984, 49985, 49986, 49987, 49988, 49989, 49990,
49991, 49992, 49993, 49994, 49995, 49996, 49997, 49998, 49999]

Execution Time in seconds:       0.2846608000108972

Execution Time in seconds:       [0.010589700017590076, 0.015800099994521588, 0.015381599980173633, 0.02351879997877404,
0.04696219999459572, 0.1135919000080321, 0.20520259998738766, 0.22601260000374168, 0.2846608000108972]
```

MERGE SORT(REVERSE SORTED INPUT):

```
    sorted_array = mergesort(merge_sort_arr)

    print("\nSorted array:",sorted_array)
    end = None
    end = timeit.default_timer()
    Execution_time=end - start
    Execution_time_arr5.append(Execution_time)
    print("\nExecution Time in seconds:\t",Execution_time)

print("\nExecution Time for each run in seconds:\t",Execution_time_arr5)
```

```
748, 49749, 49750, 49751, 49752, 49753, 49754, 49755, 49756, 49757, 49758, 49759, 49760, 49761, 49762, 49763, 49764, 4976
5, 49766, 49767, 49768, 49769, 49770, 49771, 49772, 49773, 49774, 49775, 49776, 49777, 49778, 49779, 49780, 49781, 49782,
49783, 49784, 49785, 49786, 49787, 49788, 49789, 49790, 49791, 49792, 49793, 49794, 49795, 49796, 49797, 49798, 49799, 49
800, 49801, 49802, 49803, 49804, 49805, 49806, 49807, 49808, 49809, 49810, 49811, 49812, 49813, 49814, 49815, 49816, 4981
7, 49818, 49819, 49820, 49821, 49822, 49823, 49824, 49825, 49826, 49827, 49828, 49829, 49830, 49831, 49832, 49833, 49834,
49835, 49836, 49837, 49838, 49839, 49840, 49841, 49842, 49843, 49844, 49845, 49846, 49847, 49848, 49849, 49850, 49851, 49
852, 49853, 49854, 49855, 49856, 49857, 49858, 49859, 49860, 49861, 49862, 49863, 49864, 49865, 49866, 49867, 49868, 4986
9, 49870, 49871, 49872, 49873, 49874, 49875, 49876, 49877, 49878, 49879, 49880, 49881, 49882, 49883, 49884, 49885, 49886,
49887, 49888, 49889, 49890, 49891, 49892, 49893, 49894, 49895, 49896, 49897, 49898, 49899, 49900, 49901, 49902, 49903, 49
904, 49905, 49906, 49907, 49908, 49909, 49910, 49911, 49912, 49913, 49914, 49915, 49916, 49917, 49918, 49919, 49920, 4992
1, 49922, 49923, 49924, 49925, 49926, 49927, 49928, 49929, 49930, 49931, 49932, 49933, 49934, 49935, 49936, 49937, 49938,
49939, 49940, 49941, 49942, 49943, 49944, 49945, 49946, 49947, 49948, 49949, 49950, 49951, 49952, 49953, 49954, 49955, 49
956, 49957, 49958, 49959, 49960, 49961, 49962, 49963, 49964, 49965, 49966, 49967, 49968, 49969, 49970, 49971, 49972, 4997
3, 49974, 49975, 49976, 49977, 49978, 49979, 49980, 49981, 49982, 49983, 49984, 49985, 49986, 49987, 49988, 49989, 49990,
49991, 49992, 49993, 49994, 49995, 49996, 49997, 49998, 49999]

Execution Time in seconds:       0.2530269999988377

Execution Time in seconds:       [0.004658799996832386, 0.009674500004621223, 0.01329440000699833, 0.023217800015117973,
0.04118729999754578, 0.08808760001556948, 0.14332829997874796, 0.21249259999603964, 0.2530269999988377]
```

```
    sorted_array = mergesort(merge_sort_arr)

    print("\nSorted array:",sorted_array)
    end = None
    end = timeit.default_timer()
    Execution_time=end - start
    Execution_time_arr5.append(Execution_time)
    print("\nExecution Time in seconds:\t",Execution_time)

print("\nExecution Time for each run in seconds:\t",Execution_time_arr5)
```

```
748, 49749, 49750, 49751, 49752, 49753, 49754, 49755, 49756, 49757, 49758, 49759, 49760, 49761, 49762, 49763, 49764, 4976
5, 49766, 49767, 49768, 49769, 49770, 49771, 49772, 49773, 49774, 49775, 49776, 49777, 49778, 49779, 49780, 49781, 49782,
49783, 49784, 49785, 49786, 49787, 49788, 49789, 49790, 49791, 49792, 49793, 49794, 49795, 49796, 49797, 49798, 49799, 49
800, 49801, 49802, 49803, 49804, 49805, 49806, 49807, 49808, 49809, 49810, 49811, 49812, 49813, 49814, 49815, 49816, 4981
7, 49818, 49819, 49820, 49821, 49822, 49823, 49824, 49825, 49826, 49827, 49828, 49829, 49830, 49831, 49832, 49833, 49834,
49835, 49836, 49837, 49838, 49839, 49840, 49841, 49842, 49843, 49844, 49845, 49846, 49847, 49848, 49849, 49850, 49851, 49
852, 49853, 49854, 49855, 49856, 49857, 49858, 49859, 49860, 49861, 49862, 49863, 49864, 49865, 49866, 49867, 49868, 4986
9, 49870, 49871, 49872, 49873, 49874, 49875, 49876, 49877, 49878, 49879, 49880, 49881, 49882, 49883, 49884, 49885, 49886,
49887, 49888, 49889, 49890, 49891, 49892, 49893, 49894, 49895, 49896, 49897, 49898, 49899, 49900, 49901, 49902, 49903, 49
904, 49905, 49906, 49907, 49908, 49909, 49910, 49911, 49912, 49913, 49914, 49915, 49916, 49917, 49918, 49919, 49920, 4992
1, 49922, 49923, 49924, 49925, 49926, 49927, 49928, 49929, 49930, 49931, 49932, 49933, 49934, 49935, 49936, 49937, 49938,
49939, 49940, 49941, 49942, 49943, 49944, 49945, 49946, 49947, 49948, 49949, 49950, 49951, 49952, 49953, 49954, 49955, 49
956, 49957, 49958, 49959, 49960, 49961, 49962, 49963, 49964, 49965, 49966, 49967, 49968, 49969, 49970, 49971, 49972, 4997
3, 49974, 49975, 49976, 49977, 49978, 49979, 49980, 49981, 49982, 49983, 49984, 49985, 49986, 49987, 49988, 49989, 49990,
49991, 49992, 49993, 49994, 49995, 49996, 49997, 49998, 49999]

Execution Time in seconds:       0.28511860000435263

Execution Time for each run in seconds:  [0.005350899999029934, 0.009438400011276826, 0.013936900009866804, 0.02223559998
6743182, 0.0440701000043191, 0.0888639000186231, 0.14602760001434945, 0.20563390001188964, 0.28511860000435263]
```

```
    sorted_array = mergesort(merge_sort_arr)

    print("\nSorted array:",sorted_array)
    end = None
    end = timeit.default_timer()
    Execution_time=end - start
    Execution_time_arr5.append(Execution_time)
    print("\nExecution Time in seconds:\t",Execution_time)

print("\nExecution Time for each run in seconds:\t",Execution_time_arr5)
```

```
748, 49749, 49750, 49751, 49752, 49753, 49754, 49755, 49756, 49757, 49758, 49759, 49760, 49761, 49762, 49763, 49764, 4976
5, 49766, 49767, 49768, 49769, 49770, 49771, 49772, 49773, 49774, 49775, 49776, 49777, 49778, 49779, 49780, 49781, 49782,
49783, 49784, 49785, 49786, 49787, 49788, 49789, 49790, 49791, 49792, 49793, 49794, 49795, 49796, 49797, 49798, 49799, 49
800, 49801, 49802, 49803, 49804, 49805, 49806, 49807, 49808, 49809, 49810, 49811, 49812, 49813, 49814, 49815, 49816, 4981
7, 49818, 49819, 49820, 49821, 49822, 49823, 49824, 49825, 49826, 49827, 49828, 49829, 49830, 49831, 49832, 49833, 49834,
49835, 49836, 49837, 49838, 49839, 49840, 49841, 49842, 49843, 49844, 49845, 49846, 49847, 49848, 49849, 49850, 49851, 49
852, 49853, 49854, 49855, 49856, 49857, 49858, 49859, 49860, 49861, 49862, 49863, 49864, 49865, 49866, 49867, 49868, 4986
9, 49870, 49871, 49872, 49873, 49874, 49875, 49876, 49877, 49878, 49879, 49880, 49881, 49882, 49883, 49884, 49885, 49886,
49887, 49888, 49889, 49890, 49891, 49892, 49893, 49894, 49895, 49896, 49897, 49898, 49899, 49900, 49901, 49902, 49903, 49
904, 49905, 49906, 49907, 49908, 49909, 49910, 49911, 49912, 49913, 49914, 49915, 49916, 49917, 49918, 49919, 49920, 4992
1, 49922, 49923, 49924, 49925, 49926, 49927, 49928, 49929, 49930, 49931, 49932, 49933, 49934, 49935, 49936, 49937, 49938,
49939, 49940, 49941, 49942, 49943, 49944, 49945, 49946, 49947, 49948, 49949, 49950, 49951, 49952, 49953, 49954, 49955, 49
956, 49957, 49958, 49959, 49960, 49961, 49962, 49963, 49964, 49965, 49966, 49967, 49968, 49969, 49970, 49971, 49972, 4997
3, 49974, 49975, 49976, 49977, 49978, 49979, 49980, 49981, 49982, 49983, 49984, 49985, 49986, 49987, 49988, 49989, 49990,
49991, 49992, 49993, 49994, 49995, 49996, 49997, 49998, 49999]

Execution Time in seconds:       0.294067000009818

Execution Time for each run in seconds:  [0.0048745000094641, 0.00765549999778159, 0.013353999995160848, 0.02584679998108
186, 0.04411019998951815, 0.09051830001408234, 0.15232470000046305, 0.19793170000775717, 0.294067000009818]
```

HEAP SORT(RANDOM INPUT):

```
    def main():

        bh = BinHeap()
        bh.buildHeap(heapsort_arr)

        for i in range(len(heapsort_arr)):
            print(bh.delMin())

    main()
    end = None
    end=timeit.default_timer()
    Execution_time=end - start
    Execution_time_arr6.append(Execution_time)
    print("\nExecution Time in seconds:",Execution_time)

print("\nExecution Time in seconds:\t",Execution_time_arr6)
```

```
49985
49986
49987
49988
49989
49990
49991
49992
49993
49994
49995
49996
49997
49998
49999

Execution Time in seconds: 3.096893399982946

Execution Time in seconds:       [0.026218099985271692, 0.1304921000264585, 0.1755721000081394, 0.27181669999845326, 0.62
81715999939479, 1.1370265999867115, 1.8348091999941971, 2.681126499985112, 3.096893399982946]
```

```
    def main():

        bh = BinHeap()
        bh.buildHeap(heapsort_arr)

        for i in range(len(heapsort_arr)):
            print(bh.delMin())

    main()
    end = None
    end=timeit.default_timer()
    Execution_time=end - start
    Execution_time_arr6.append(Execution_time)
    print("\nExecution Time in seconds:",Execution_time)

print("\nExecution Time in seconds:\t",Execution_time_arr6)
```

```
49985
49986
49987
49988
49989
49990
49991
49992
49993
49994
49995
49996
49997
49998
49999

Execution Time in seconds: 3.1836351999954786

Execution Time in seconds:       [0.03996169997844845, 0.12526129998150282, 0.19823099998757243, 0.26935849999426864, 0.5
916252000024542, 1.2106986999860965, 1.8721782000211533, 2.5133238999987952, 3.1836351999954786]
```

```
    def main():

        bh = BinHeap()
        bh.buildHeap(heapsort_arr)

        for i in range(len(heapsort_arr)):
            print(bh.delMin())

    main()
    end = None
    end=timeit.default_timer()
    Execution_time=end - start
    Execution_time_arr6.append(Execution_time)
    print("\nExecution Time in seconds:",Execution_time)

print("\nExecution Time in seconds:\t",Execution_time_arr6)
```

```
49985
49986
49987
49988
49989
49990
49991
49992
49993
49994
49995
49996
49997
49998
49999

Execution Time in seconds: 3.2560308999964036

Execution Time in seconds:       [0.037408099975436926, 0.1235826000047382, 0.22625750000588596, 0.2665631999843754, 0.54
9625699990429, 1.210730999999214, 1.8748923999955878, 2.568216099985875, 3.2560308999964036]
```

HEAP SORT(SORTED INPUT):

```
    def main():

        bh = BinHeap()
        bh.buildHeap(heapsort_arr)

        for i in range(len(heapsort_arr)):
            print(bh.delMin())

    main()
    end = None
    end=timeit.default_timer()
    Execution_time=end - start
    Execution_time_arr7.append(Execution_time)
    print("\nExecution Time in seconds:",Execution_time)

print("\nExecution Time in seconds:\t",Execution_time_arr7)
```

```
49985
49986
49987
49988
49989
49990
49991
49992
49993
49994
49995
49996
49997
49998
49999

Execution Time in seconds: 4.715326700010337

Execution Time in seconds:       [0.04417839998495765, 0.13725699999486096, 0.23493919998873025, 0.37690900001325645, 0.9
851760000165086, 2.189317199983634, 3.0201541999995243, 3.646095800009789, 4.715326700010337]
```

```
    def main():

        bh = BinHeap()
        bh.buildHeap(heapsort_arr)

        for i in range(len(heapsort_arr)):
            print(bh.delMin())

    main()
    end = None
    end=timeit.default_timer()
    Execution_time=end - start
    Execution_time_arr7.append(Execution_time)
    print("\nExecution Time in seconds:",Execution_time)

print("\nExecution Time in seconds:\t",Execution_time_arr7)
```

```
49985
49986
49987
49988
49989
49990
49991
49992
49993
49994
49995
49996
49997
49998
49999

Execution Time in seconds: 4.889465500018559

Execution Time in seconds:       [0.032009500020649284, 0.1406862000003457, 0.1774808999907691, 0.28778610000154004, 0.61
28502000065055, 1.1713214000046719, 1.863414899999043, 4.17181209998671, 4.889465500018559]
```

```
    def main():

        bh = BinHeap()
        bh.buildHeap(heapsort_arr)

        for i in range(len(heapsort_arr)):
            print(bh.delMin())

    main()
    end = None
    end=timeit.default_timer()
    Execution_time=end - start
    Execution_time_arr7.append(Execution_time)
    print("\nExecution Time in seconds:",Execution_time)

print("\nExecution Time for each run in seconds:\t",Execution_time_arr7)
```

```
49985
49986
49987
49988
49989
49990
49991
49992
49993
49994
49995
49996
49997
49998
49999

Execution Time in seconds: 4.450692699989304

Execution Time for each run in seconds:  [0.03607730002840981, 0.13265519999549724, 0.18327800001134165, 0.33718599998974
24, 0.6228013000218198, 1.5684263999864925, 3.422389600018505, 3.688021799986018, 4.450692699989304]
```

HEAP SORT(REVERSE SORTED INPUT)

```
    def main():

        bh = BinHeap()
        bh.buildHeap(heapsort_arr)

        for i in range(len(heapsort_arr)):
            print(bh.delMin())

    main()
    end = None
    end=timeit.default_timer()
    Execution_time=end - start
    Execution_time_arr8.append(Execution_time)
    print("\nExecution Time in seconds:",Execution_time)

print("\nExecution Time in seconds:\t",Execution_time_arr8)
```

```
49985
49986
49987
49988
49989
49990
49991
49992
49993
49994
49995
49996
49997
49998
49999

Execution Time in seconds: 3.1955062999913935

Execution Time in seconds:       [0.04428939998615533, 0.12334950000513345, 0.17005159999825992, 0.27562749999924563, 0.6
299333999922965, 1.2258386999892537, 1.9188545999932103, 2.503751899988856, 3.1955062999913935]
```

```
    def main():

        bh = BinHeap()
        bh.buildHeap(heapsort_arr)

        for i in range(len(heapsort_arr)):
            print(bh.delMin())

    main()
    end = None
    end=timeit.default_timer()
    Execution_time=end - start
    Execution_time_arr8.append(Execution_time)
    print("\nExecution Time in seconds:",Execution_time)

print("\nExecution Time in seconds:\t",Execution_time_arr8)
```

```
49985
49986
49987
49988
49989
49990
49991
49992
49993
49994
49995
49996
49997
49998
49999

Execution Time in seconds: 3.08504589999211

Execution Time in seconds:      [0.04051269998308271, 0.12401489997864701, 0.18702379998285323, 0.2767986000108067, 0.58
82303999969736, 1.2533792000031099, 1.908065199997509, 2.5751850999949966, 3.08504589999211]
```

```
    def main():

        bh = BinHeap()
        bh.buildHeap(heapsort_arr)

        for i in range(len(heapsort_arr)):
            print(bh.delMin())

    main()
    end = None
    end=timeit.default_timer()
    Execution_time=end - start
    Execution_time_arr8.append(Execution_time)
    print("\nExecution Time in seconds:",Execution_time)

print("\nExecution Time for each run in seconds:\t",Execution_time_arr8)
```

```
49985
49986
49987
49988
49989
49990
49991
49992
49993
49994
49995
49996
49997
49998
49999

Execution Time in seconds: 3.148024100024486

Execution Time in seconds:      [0.038527099997736514, 0.12864740000804886, 0.18856230002711527, 0.28011330001754686, 0.
621934599999804, 1.2398685999796726, 1.8495222000055946, 2.4552828000159934, 3.148024100024486]
```

INPLACE QUICK SORT(RANDOM INPUT)

```
    quicksort_inplace(quicksort_arr, 0, len(quicksort_arr)-1)
    print ("\nSorted array", quicksort_arr)
    end=None
    end = timeit.default_timer()
    Execution_time=end - start
    Execution_time_arr9.append(Execution_time)
    print("\nExecution Time in seconds:",Execution_time)

print("\nExecution Time for each run in seconds:\t",Execution_time_arr9)
```

```
413, 27510, 27558, 27600, 27608, 27720, 27845, 27908, 27979, 27993, 28095, 28173, 28258, 28284, 28465, 28523, 28584, 2866
8, 28732, 29138, 29563, 29904, 29928, 30173, 30200, 30201, 30418, 30455, 30556, 30751, 30819, 30982, 30991, 31034, 31067,
31218, 31273, 31283, 31525, 31583, 31588, 31641, 31687, 31948, 32080, 32112, 32432, 32515, 32768, 32906, 32976, 33123, 33
228, 33242, 33432, 33510, 33573, 33777, 33973, 34014, 34057, 34083, 34152, 34287, 34404, 34550, 34554, 34668, 34709, 3474
5, 34863, 35002, 35037, 35099, 35116, 35192, 35453, 35476, 35505, 35642, 35666, 35941, 36020, 36052, 36096, 36100, 36262,
36337, 36406, 36626, 36639, 36651, 36938, 36952, 36960, 36983, 37141, 37246, 37250, 37297, 37306, 37414, 37483, 37587, 37
619, 37672, 37808, 37811, 37950, 38008, 38066, 38289, 38343, 38428, 38444, 38519, 38579, 38614, 38625, 38632, 38659, 3873
8, 38839, 38841, 38977, 39034, 39036, 39306, 39369, 39416, 39578, 39612, 39648, 39688, 39702, 39902, 40007, 40118, 40274,
40356, 40507, 40588, 40640, 40806, 40824, 41007, 41068, 41113, 41120, 41154, 41196, 41430, 41482, 41519, 41561, 41568, 41
606, 41725, 41861, 41881, 42070, 42197, 42334, 42396, 42568, 42613, 42616, 42842, 42959, 43201, 43403, 43443, 43449, 4374
9, 43992, 44001, 44042, 44140, 44232, 44303, 44437, 44517, 44628, 44694, 44809, 45202, 45256, 45462, 45490, 45558, 45584,
45635, 45748, 45784, 46063, 46330, 46708, 46780, 46801, 46868, 46898, 46926, 46961, 46999, 47006, 47083, 47108, 47206, 47
385, 47718, 47941, 48086, 48180, 48367, 48578, 48632, 48843, 48858, 49021, 49084, 49373, 49577, 49683, 49691, 49810, 4991
4]

Execution Time in seconds: 0.0026823000000604225

Execution Time for each run in seconds:  [0.000891700000011042, 0.000519199999985176, 0.0005515000000286818, 0.0003976000
000420754, 0.0006596999999146647, 0.006241400000021713, 0.0020253000000138854, 0.0029460000000653963, 0.00268230000006042
25]
```

```
    end = timeit.default_timer()
    Execution_time=end - start
    Execution_time_arr10.append(Execution_time)
    print("\nExecution Time in seconds:",Execution_time)

print("\nExecution Time for each run in seconds:\t",Execution_time_arr10)
```

```
25309, 25402, 25472, 25535, 25554, 25884, 26026, 26232, 26710, 26734, 26838, 26900, 27083, 27115, 27210, 27211, 27322, 27
413, 27510, 27558, 27600, 27608, 27720, 27845, 27908, 27979, 27993, 28095, 28173, 28258, 28284, 28465, 28523, 28584, 2866
8, 28732, 29138, 29563, 29904, 29928, 30173, 30200, 30201, 30418, 30455, 30556, 30751, 30819, 30982, 30991, 31034, 31067,
31218, 31273, 31283, 31525, 31583, 31588, 31641, 31687, 31948, 32080, 32112, 32432, 32515, 32768, 32906, 32976, 33123, 33
228, 33242, 33432, 33510, 33573, 33777, 33973, 34014, 34057, 34083, 34152, 34287, 34404, 34550, 34554, 34668, 34709, 3474
5, 34863, 35002, 35037, 35099, 35116, 35192, 35453, 35476, 35505, 35642, 35666, 35941, 36020, 36052, 36096, 36100, 36262,
36337, 36406, 36626, 36639, 36651, 36938, 36952, 36960, 36983, 37141, 37246, 37250, 37297, 37306, 37414, 37483, 37587, 37
619, 37672, 37808, 37811, 37950, 38008, 38066, 38289, 38343, 38428, 38444, 38519, 38579, 38614, 38625, 38632, 38659, 3873
8, 38839, 38841, 38977, 39034, 39036, 39306, 39369, 39416, 39578, 39612, 39648, 39688, 39702, 39902, 40007, 40118, 40274,
40356, 40507, 40588, 40640, 40806, 40824, 41007, 41068, 41113, 41120, 41154, 41196, 41430, 41482, 41519, 41561, 41568, 41
606, 41725, 41861, 41881, 42070, 42197, 42334, 42396, 42568, 42613, 42616, 42842, 42959, 43201, 43403, 43443, 43449, 4374
9, 43992, 44001, 44042, 44140, 44232, 44303, 44437, 44517, 44628, 44694, 44809, 45202, 45256, 45462, 45490, 45558, 45584,
45635, 45748, 45784, 46063, 46330, 46708, 46780, 46801, 46868, 46898, 46926, 46961, 46999, 47006, 47083, 47108, 47206, 47
385, 47718, 47941, 48086, 48180, 48367, 48578, 48632, 48843, 48858, 49021, 49084, 49373, 49577, 49683, 49691, 49810, 4991
4]

Execution Time in seconds: 0.03564130000006571

Execution Time for each run in seconds:  [0.00029230000006919, 0.00022799999999278953, 0.0003227999999353415, 0.000586500
0000540022, 0.003106800000068688, 0.008050200000070618, 0.016822499999989304, 0.031120200000032128, 0.03564130000006571]
```

```
    quicksort_inplace(quicksort_arr, 0, len(quicksort_arr)-1)
    print ("\nSorted array", quicksort_arr)
    end=None
    end = timeit.default_timer()
    Execution_time=end - start
    Execution_time_arr11.append(Execution_time)
    print("\nExecution Time in seconds:",Execution_time)

print("\nExecution Time for each run in seconds:\t",Execution_time_arr11)
```

```
25309, 25402, 25472, 25535, 25554, 25884, 26026, 26232, 26710, 26734, 26838, 26900, 27083, 27115, 27210, 27211, 27322, 27
413, 27510, 27558, 27600, 27608, 27720, 27845, 27908, 27979, 27993, 28095, 28173, 28258, 28284, 28465, 28523, 28584, 2866
8, 28732, 29138, 29563, 29904, 29928, 30173, 30200, 30201, 30418, 30455, 30556, 30751, 30819, 30982, 30991, 31034, 31067,
31218, 31273, 31283, 31525, 31583, 31588, 31641, 31687, 31948, 32080, 32112, 32432, 32515, 32768, 32906, 32976, 33123, 33
228, 33242, 33432, 33510, 33573, 33777, 33973, 34014, 34057, 34083, 34152, 34287, 34404, 34550, 34554, 34668, 34709, 3474
5, 34863, 35002, 35037, 35099, 35116, 35192, 35453, 35476, 35505, 35642, 35666, 35941, 36020, 36052, 36096, 36100, 36262,
36337, 36406, 36626, 36639, 36651, 36938, 36952, 36960, 36983, 37141, 37246, 37250, 37297, 37306, 37414, 37483, 37587, 37
619, 37672, 37808, 37811, 37950, 38008, 38066, 38289, 38343, 38428, 38444, 38519, 38579, 38614, 38625, 38632, 38659, 3873
8, 38839, 38841, 38977, 39034, 39036, 39306, 39369, 39416, 39578, 39612, 39648, 39688, 39702, 39902, 40007, 40118, 40274,
40356, 40507, 40588, 40640, 40806, 40824, 41007, 41068, 41113, 41120, 41154, 41196, 41430, 41482, 41519, 41561, 41568, 41
606, 41725, 41861, 41881, 42070, 42197, 42334, 42396, 42568, 42613, 42616, 42842, 42959, 43201, 43403, 43443, 43449, 4374
9, 43992, 44001, 44042, 44140, 44232, 44303, 44437, 44517, 44628, 44694, 44809, 45202, 45256, 45462, 45490, 45558, 45584,
45635, 45748, 45784, 46063, 46330, 46708, 46780, 46801, 46868, 46898, 46926, 46961, 46999, 47006, 47083, 47108, 47206, 47
385, 47718, 47941, 48086, 48180, 48367, 48578, 48632, 48843, 48858, 49021, 49084, 49373, 49577, 49683, 49691, 49810, 4991
4]

Execution Time in seconds: 0.03145710000001145

Execution Time for each run in seconds:  [0.0008486000000402782, 0.0008328000000119573, 0.004415099999960148, 0.001803799
9999478416, 0.002264599999989514, 0.010448699999983546, 0.015787699999918914, 0.023040000000037253, 0.03145710000001145]
```

MODIFIED QUICK SORT(RANDOM INPUT)

```
    def quicksort_m(array, leftindex, rightindex):
        if(len(quicksort_marr)<=15):
            insertion_sort(quicksort_marr)
        elif(len(quicksort_marr)>15):
            if leftindex < rightindex:
                newpivotindex = partition_m(array, leftindex, rightindex)
                quicksort_m(array, leftindex, newpivotindex)
                quicksort_m(array, newpivotindex + 1, rightindex)

    quicksort_m(quicksort_marr, 0, len(quicksort_marr))
    print("\nQuick sort",quicksort_marr)
    end=None
    end = timeit.default_timer()
    Execution_time=end - start
    Execution_time_arr12.append(Execution_time)
    print("\nExecution Time in seconds:",Execution_time)
print("\nExecution Time in seconds:\t",Execution_time_arr12)
```

```
749, 49750, 49751, 49752, 49753, 49754, 49755, 49756, 49757, 49758, 49759, 49760, 49761, 49762, 49763, 49764, 49765, 4976
6, 49767, 49768, 49769, 49770, 49771, 49772, 49773, 49774, 49775, 49776, 49777, 49778, 49779, 49780, 49781, 49782, 49783,
49784, 49785, 49786, 49787, 49788, 49789, 49790, 49791, 49792, 49793, 49794, 49795, 49796, 49797, 49798, 49799, 49800, 49
801, 49802, 49803, 49804, 49805, 49806, 49807, 49808, 49809, 49810, 49811, 49812, 49813, 49814, 49815, 49816, 49817, 4981
8, 49819, 49820, 49821, 49822, 49823, 49824, 49825, 49826, 49827, 49828, 49829, 49830, 49831, 49832, 49833, 49834, 49835,
49836, 49837, 49838, 49839, 49840, 49841, 49842, 49843, 49844, 49845, 49846, 49847, 49848, 49849, 49850, 49851, 49852, 49
853, 49854, 49855, 49856, 49857, 49858, 49859, 49860, 49861, 49862, 49863, 49864, 49865, 49866, 49867, 49868, 49869, 4987
0, 49871, 49872, 49873, 49874, 49875, 49876, 49877, 49878, 49879, 49880, 49881, 49882, 49883, 49884, 49885, 49886, 49887,
49888, 49889, 49890, 49891, 49892, 49893, 49894, 49895, 49896, 49897, 49898, 49899, 49900, 49901, 49902, 49903, 49904, 49
905, 49906, 49907, 49908, 49909, 49910, 49911, 49912, 49913, 49914, 49915, 49916, 49917, 49918, 49919, 49920, 49921, 4992
2, 49923, 49924, 49925, 49926, 49927, 49928, 49929, 49930, 49931, 49932, 49933, 49934, 49935, 49936, 49937, 49938, 49939,
49940, 49941, 49942, 49943, 49944, 49945, 49946, 49947, 49948, 49949, 49950, 49951, 49952, 49953, 49954, 49955, 49956, 49
957, 49958, 49959, 49960, 49961, 49962, 49963, 49964, 49965, 49966, 49967, 49968, 49969, 49970, 49971, 49972, 49973, 4997
4, 49975, 49976, 49977, 49978, 49979, 49980, 49981, 49982, 49983, 49984, 49985, 49986, 49987, 49988, 49989, 49990, 49991,
49992, 49993, 49994, 49995, 49996, 49997, 49998, 49999]

Execution Time in seconds: 12.371878399999986

Execution Time in seconds:      [0.01003829999999084, 0.02862479999998868, 0.05180970000000684, 0.14511340000001383, 0.5
335460999999952, 1.9776447999999789, 4.674305000000004, 8.048928399999994, 12.371878399999986]
```

```
    def quicksort_m(array, leftindex, rightindex):
        if(len(quicksort_marr)<=15):
            insertion_sort(quicksort_marr)
        elif(len(quicksort_marr)>15):
             if leftindex < rightindex:
                newpivotindex = partition_m(array, leftindex, rightindex)
                quicksort_m(array, leftindex, newpivotindex)
                quicksort_m(array, newpivotindex + 1, rightindex)

    quicksort_m(quicksort_marr, 0, len(quicksort_marr))
    print("\nQuick sort",quicksort_marr)
    end=None
    end = timeit.default_timer()
    Execution_time=end - start
    Execution_time_arr12.append(Execution_time)
    print("\nExecution Time in seconds:",Execution_time)
print("\nExecution Time in seconds:\t",Execution_time_arr12)
```

```
749, 49750, 49751, 49752, 49753, 49754, 49755, 49756, 49757, 49758, 49759, 49760, 49761, 49762, 49763, 49764, 49765, 4976
6, 49767, 49768, 49769, 49770, 49771, 49772, 49773, 49774, 49775, 49776, 49777, 49778, 49779, 49780, 49781, 49782, 49783,
49784, 49785, 49786, 49787, 49788, 49789, 49790, 49791, 49792, 49793, 49794, 49795, 49796, 49797, 49798, 49799, 49800, 49
801, 49802, 49803, 49804, 49805, 49806, 49807, 49808, 49809, 49810, 49811, 49812, 49813, 49814, 49815, 49816, 49817, 4981
8, 49819, 49820, 49821, 49822, 49823, 49824, 49825, 49826, 49827, 49828, 49829, 49830, 49831, 49832, 49833, 49834, 49835,
49836, 49837, 49838, 49839, 49840, 49841, 49842, 49843, 49844, 49845, 49846, 49847, 49848, 49849, 49850, 49851, 49852, 49
853, 49854, 49855, 49856, 49857, 49858, 49859, 49860, 49861, 49862, 49863, 49864, 49865, 49866, 49867, 49868, 49869, 4987
0, 49871, 49872, 49873, 49874, 49875, 49876, 49877, 49878, 49879, 49880, 49881, 49882, 49883, 49884, 49885, 49886, 49887,
49888, 49889, 49890, 49891, 49892, 49893, 49894, 49895, 49896, 49897, 49898, 49899, 49900, 49901, 49902, 49903, 49904, 49
905, 49906, 49907, 49908, 49909, 49910, 49911, 49912, 49913, 49914, 49915, 49916, 49917, 49918, 49919, 49920, 49921, 4992
2, 49923, 49924, 49925, 49926, 49927, 49928, 49929, 49930, 49931, 49932, 49933, 49934, 49935, 49936, 49937, 49938, 49939,
49940, 49941, 49942, 49943, 49944, 49945, 49946, 49947, 49948, 49949, 49950, 49951, 49952, 49953, 49954, 49955, 49956, 49
957, 49958, 49959, 49960, 49961, 49962, 49963, 49964, 49965, 49966, 49967, 49968, 49969, 49970, 49971, 49972, 49973, 4997
4, 49975, 49976, 49977, 49978, 49979, 49980, 49981, 49982, 49983, 49984, 49985, 49986, 49987, 49988, 49989, 49990, 49991,
49992, 49993, 49994, 49995, 49996, 49997, 49998, 49999]

Execution Time in seconds: 11.99906820000001

Execution Time in seconds:       [0.009757399999955396, 0.027999099999988175, 0.05808329999996431, 0.14184620000003179,
0.5780613000000017, 2.012463000000025, 4.553390399999955, 8.165913999999987, 11.99906820000001]
```

```
    def quicksort_m(array, leftindex, rightindex):
        if(len(quicksort_marr)<=15):
            insertion_sort(quicksort_marr)
        elif(len(quicksort_marr)>15):
             if leftindex < rightindex:
                newpivotindex = partition_m(array, leftindex, rightindex)
                quicksort_m(array, leftindex, newpivotindex)
                quicksort_m(array, newpivotindex + 1, rightindex)

    quicksort_m(quicksort_marr, 0, len(quicksort_marr))
    print("\nQuick sort",quicksort_marr)
    end=None
    end = timeit.default_timer()
    Execution_time=end - start
    Execution_time_arr12.append(Execution_time)
    print("\nExecution Time in seconds:",Execution_time)
print("\nExecution Time in seconds:\t",Execution_time_arr12)
```

```
749, 49750, 49751, 49752, 49753, 49754, 49755, 49756, 49757, 49758, 49759, 49760, 49761, 49762, 49763, 49764, 49765, 4976
6, 49767, 49768, 49769, 49770, 49771, 49772, 49773, 49774, 49775, 49776, 49777, 49778, 49779, 49780, 49781, 49782, 49783,
49784, 49785, 49786, 49787, 49788, 49789, 49790, 49791, 49792, 49793, 49794, 49795, 49796, 49797, 49798, 49799, 49800, 49
801, 49802, 49803, 49804, 49805, 49806, 49807, 49808, 49809, 49810, 49811, 49812, 49813, 49814, 49815, 49816, 49817, 4981
8, 49819, 49820, 49821, 49822, 49823, 49824, 49825, 49826, 49827, 49828, 49829, 49830, 49831, 49832, 49833, 49834, 49835,
49836, 49837, 49838, 49839, 49840, 49841, 49842, 49843, 49844, 49845, 49846, 49847, 49848, 49849, 49850, 49851, 49852, 49
853, 49854, 49855, 49856, 49857, 49858, 49859, 49860, 49861, 49862, 49863, 49864, 49865, 49866, 49867, 49868, 49869, 4987
0, 49871, 49872, 49873, 49874, 49875, 49876, 49877, 49878, 49879, 49880, 49881, 49882, 49883, 49884, 49885, 49886, 49887,
49888, 49889, 49890, 49891, 49892, 49893, 49894, 49895, 49896, 49897, 49898, 49899, 49900, 49901, 49902, 49903, 49904, 49
905, 49906, 49907, 49908, 49909, 49910, 49911, 49912, 49913, 49914, 49915, 49916, 49917, 49918, 49919, 49920, 49921, 4992
2, 49923, 49924, 49925, 49926, 49927, 49928, 49929, 49930, 49931, 49932, 49933, 49934, 49935, 49936, 49937, 49938, 49939,
49940, 49941, 49942, 49943, 49944, 49945, 49946, 49947, 49948, 49949, 49950, 49951, 49952, 49953, 49954, 49955, 49956, 49
957, 49958, 49959, 49960, 49961, 49962, 49963, 49964, 49965, 49966, 49967, 49968, 49969, 49970, 49971, 49972, 49973, 4997
4, 49975, 49976, 49977, 49978, 49979, 49980, 49981, 49982, 49983, 49984, 49985, 49986, 49987, 49988, 49989, 49990, 49991,
49992, 49993, 49994, 49995, 49996, 49997, 49998, 49999]

Execution Time in seconds: 12.032458399999996

Execution Time in seconds:       [0.008310600000015711, 0.028273499999954765, 0.05805619999995315, 0.14959900000002335,
0.5641751999999656, 2.0644015999999965, 4.3769603000000075, 7.603804200000013, 12.032458399999996]
```

MODIFIED QUICK SORT(SORTED INPUT)

```
    def quicksort_median(array, leftindex, rightindex):
        if(len(quicksort_marr)<=15):
            insertion_sort(quicksort_marr)
        elif(len(quicksort_marr)>15):
             if leftindex < rightindex:
                newpivotindex = partition_median(array, leftindex, rightindex)
                quicksort_median(array, leftindex, newpivotindex)
                quicksort_median(array, newpivotindex + 1, rightindex)

    quicksort_median(quicksort_marr, 0, len(quicksort_marr))
    print("\nQuick sort",quicksort_marr)
    end=None
    end = timeit.default_timer()
    Execution_time=end - start
    Execution_time_arr13.append(Execution_time)
    print("\nExecution Time in seconds:",Execution_time)
print("\nExecution Time in seconds:\t",Execution_time_arr13)
```

```
749, 49750, 49751, 49752, 49753, 49754, 49755, 49756, 49757, 49758, 49759, 49760, 49761, 49762, 49763, 49764, 49765, 4976
6, 49767, 49768, 49769, 49770, 49771, 49772, 49773, 49774, 49775, 49776, 49777, 49778, 49779, 49780, 49781, 49782, 49783,
49784, 49785, 49786, 49787, 49788, 49789, 49790, 49791, 49792, 49793, 49794, 49795, 49796, 49797, 49798, 49799, 49800, 49
801, 49802, 49803, 49804, 49805, 49806, 49807, 49808, 49809, 49810, 49811, 49812, 49813, 49814, 49815, 49816, 49817, 4981
8, 49819, 49820, 49821, 49822, 49823, 49824, 49825, 49826, 49827, 49828, 49829, 49830, 49831, 49832, 49833, 49834, 49835,
49836, 49837, 49838, 49839, 49840, 49841, 49842, 49843, 49844, 49845, 49846, 49847, 49848, 49849, 49850, 49851, 49852, 49
853, 49854, 49855, 49856, 49857, 49858, 49859, 49860, 49861, 49862, 49863, 49864, 49865, 49866, 49867, 49868, 49869, 4987
0, 49871, 49872, 49873, 49874, 49875, 49876, 49877, 49878, 49879, 49880, 49881, 49882, 49883, 49884, 49885, 49886, 49887,
49888, 49889, 49890, 49891, 49892, 49893, 49894, 49895, 49896, 49897, 49898, 49899, 49900, 49901, 49902, 49903, 49904, 49
905, 49906, 49907, 49908, 49909, 49910, 49911, 49912, 49913, 49914, 49915, 49916, 49917, 49918, 49919, 49920, 49921, 4992
2, 49923, 49924, 49925, 49926, 49927, 49928, 49929, 49930, 49931, 49932, 49933, 49934, 49935, 49936, 49937, 49938, 49939,
49940, 49941, 49942, 49943, 49944, 49945, 49946, 49947, 49948, 49949, 49950, 49951, 49952, 49953, 49954, 49955, 49956, 49
957, 49958, 49959, 49960, 49961, 49962, 49963, 49964, 49965, 49966, 49967, 49968, 49969, 49970, 49971, 49972, 49973, 4997
4, 49975, 49976, 49977, 49978, 49979, 49980, 49981, 49982, 49983, 49984, 49985, 49986, 49987, 49988, 49989, 49990, 49991,
49992, 49993, 49994, 49995, 49996, 49997, 49998, 49999]

Execution Time in seconds: 11.763224800000046

Execution Time in seconds:      [0.009466799999984232, 0.02666519999996808, 0.05321920000000091, 0.1364025999999967, 0.5
827069999999708, 2.026481799999999, 4.346364899999969, 7.642523399999959, 11.763224800000046]
```

```
    def quicksort_median(array, leftindex, rightindex):
        if(len(quicksort_marr)<=15):
            insertion_sort(quicksort_marr)
        elif(len(quicksort_marr)>15):
             if leftindex < rightindex:
                newpivotindex = partition_median(array, leftindex, rightindex)
                quicksort_median(array, leftindex, newpivotindex)
                quicksort_median(array, newpivotindex + 1, rightindex)

    quicksort_median(quicksort_marr, 0, len(quicksort_marr))
    print("\nQuick sort",quicksort_marr)
    end=None
    end = timeit.default_timer()
    Execution_time=end - start
    Execution_time_arr13.append(Execution_time)
    print("\nExecution Time in seconds:",Execution_time)
print("\nExecution Time in seconds:\t",Execution_time_arr13)
```

```
749, 49750, 49751, 49752, 49753, 49754, 49755, 49756, 49757, 49758, 49759, 49760, 49761, 49762, 49763, 49764, 49765, 4976
6, 49767, 49768, 49769, 49770, 49771, 49772, 49773, 49774, 49775, 49776, 49777, 49778, 49779, 49780, 49781, 49782, 49783,
49784, 49785, 49786, 49787, 49788, 49789, 49790, 49791, 49792, 49793, 49794, 49795, 49796, 49797, 49798, 49799, 49800, 49
801, 49802, 49803, 49804, 49805, 49806, 49807, 49808, 49809, 49810, 49811, 49812, 49813, 49814, 49815, 49816, 49817, 4981
8, 49819, 49820, 49821, 49822, 49823, 49824, 49825, 49826, 49827, 49828, 49829, 49830, 49831, 49832, 49833, 49834, 49835,
49836, 49837, 49838, 49839, 49840, 49841, 49842, 49843, 49844, 49845, 49846, 49847, 49848, 49849, 49850, 49851, 49852, 49
853, 49854, 49855, 49856, 49857, 49858, 49859, 49860, 49861, 49862, 49863, 49864, 49865, 49866, 49867, 49868, 49869, 4987
0, 49871, 49872, 49873, 49874, 49875, 49876, 49877, 49878, 49879, 49880, 49881, 49882, 49883, 49884, 49885, 49886, 49887,
49888, 49889, 49890, 49891, 49892, 49893, 49894, 49895, 49896, 49897, 49898, 49899, 49900, 49901, 49902, 49903, 49904, 49
905, 49906, 49907, 49908, 49909, 49910, 49911, 49912, 49913, 49914, 49915, 49916, 49917, 49918, 49919, 49920, 49921, 4992
2, 49923, 49924, 49925, 49926, 49927, 49928, 49929, 49930, 49931, 49932, 49933, 49934, 49935, 49936, 49937, 49938, 49939,
49940, 49941, 49942, 49943, 49944, 49945, 49946, 49947, 49948, 49949, 49950, 49951, 49952, 49953, 49954, 49955, 49956, 49
957, 49958, 49959, 49960, 49961, 49962, 49963, 49964, 49965, 49966, 49967, 49968, 49969, 49970, 49971, 49972, 49973, 4997
4, 49975, 49976, 49977, 49978, 49979, 49980, 49981, 49982, 49983, 49984, 49985, 49986, 49987, 49988, 49989, 49990, 49991,
49992, 49993, 49994, 49995, 49996, 49997, 49998, 49999]

Execution Time in seconds: 11.927526700000044

Execution Time in seconds:      [0.008345800000029158, 0.02783620000002429, 0.055927199999928234, 0.13483959999996387,
0.5633678000000373, 1.9610837000000174, 4.308261799999968, 7.997615399999972, 11.927526700000044]
```

```
    def quicksort_median(array, leftindex, rightindex):
        if(len(quicksort_marr)<=15):
            insertion_sort(quicksort_marr)
        elif(len(quicksort_marr)>15):
             if leftindex < rightindex:
                newpivotindex = partition_median(array, leftindex, rightindex)
                quicksort_median(array, leftindex, newpivotindex)
                quicksort_median(array, newpivotindex + 1, rightindex)

    quicksort_median(quicksort_marr, 0, len(quicksort_marr))
    print("\nQuick sort",quicksort_marr)
    end=None
    end = timeit.default_timer()
    Execution_time=end - start
    Execution_time_arr13.append(Execution_time)
    print("\nExecution Time in seconds:",Execution_time)
print("\nExecution Time in seconds:\t",Execution_time_arr13)
```

```
749, 49750, 49751, 49752, 49753, 49754, 49755, 49756, 49757, 49758, 49759, 49760, 49761, 49762, 49763, 49764, 49765, 4976
6, 49767, 49768, 49769, 49770, 49771, 49772, 49773, 49774, 49775, 49776, 49777, 49778, 49779, 49780, 49781, 49782, 49783,
49784, 49785, 49786, 49787, 49788, 49789, 49790, 49791, 49792, 49793, 49794, 49795, 49796, 49797, 49798, 49799, 49800, 49
801, 49802, 49803, 49804, 49805, 49806, 49807, 49808, 49809, 49810, 49811, 49812, 49813, 49814, 49815, 49816, 49817, 4981
8, 49819, 49820, 49821, 49822, 49823, 49824, 49825, 49826, 49827, 49828, 49829, 49830, 49831, 49832, 49833, 49834, 49835,
49836, 49837, 49838, 49839, 49840, 49841, 49842, 49843, 49844, 49845, 49846, 49847, 49848, 49849, 49850, 49851, 49852, 49
853, 49854, 49855, 49856, 49857, 49858, 49859, 49860, 49861, 49862, 49863, 49864, 49865, 49866, 49867, 49868, 49869, 4987
0, 49871, 49872, 49873, 49874, 49875, 49876, 49877, 49878, 49879, 49880, 49881, 49882, 49883, 49884, 49885, 49886, 49887,
49888, 49889, 49890, 49891, 49892, 49893, 49894, 49895, 49896, 49897, 49898, 49899, 49900, 49901, 49902, 49903, 49904, 49
905, 49906, 49907, 49908, 49909, 49910, 49911, 49912, 49913, 49914, 49915, 49916, 49917, 49918, 49919, 49920, 49921, 4992
2, 49923, 49924, 49925, 49926, 49927, 49928, 49929, 49930, 49931, 49932, 49933, 49934, 49935, 49936, 49937, 49938, 49939,
49940, 49941, 49942, 49943, 49944, 49945, 49946, 49947, 49948, 49949, 49950, 49951, 49952, 49953, 49954, 49955, 49956, 49
957, 49958, 49959, 49960, 49961, 49962, 49963, 49964, 49965, 49966, 49967, 49968, 49969, 49970, 49971, 49972, 49973, 4997
4, 49975, 49976, 49977, 49978, 49979, 49980, 49981, 49982, 49983, 49984, 49985, 49986, 49987, 49988, 49989, 49990, 49991,
49992, 49993, 49994, 49995, 49996, 49997, 49998, 49999]

Execution Time in seconds: 11.854548499999964

Execution Time in seconds:      [0.007898700000055214, 0.027466599999911523, 0.061128400000029615, 0.14124409999999443,
0.5327254999999695, 1.9477735000000393, 4.305899499999896, 7.568081100000086, 11.854548499999964]
```

MODIFIED QUICK SORT(REVERSELY SORTED INPUT)

```
    def quicksort_median(array, leftindex, rightindex):
        if(len(quicksort_marr)<=15):
            insertion_sort(quicksort_marr)
        elif(len(quicksort_marr)>15):
             if leftindex < rightindex:
                newpivotindex = partition_median(array, leftindex, rightindex)
                quicksort_median(array, leftindex, newpivotindex)
                quicksort_median(array, newpivotindex + 1, rightindex)

    quicksort_median(quicksort_marr, 0, len(quicksort_marr))
    print("\nQuick sort",quicksort_marr)
    end=None
    end = timeit.default_timer()
    Execution_time=end - start
    Execution_time_arr14.append(Execution_time)
    print("\nExecution Time in seconds:",Execution_time)
print("\nExecution Time in seconds:\t",Execution_time_arr14)
```

```
749, 49750, 49751, 49752, 49753, 49754, 49755, 49756, 49757, 49758, 49759, 49760, 49761, 49762, 49763, 49764, 49765, 4976
6, 49767, 49768, 49769, 49770, 49771, 49772, 49773, 49774, 49775, 49776, 49777, 49778, 49779, 49780, 49781, 49782, 49783,
49784, 49785, 49786, 49787, 49788, 49789, 49790, 49791, 49792, 49793, 49794, 49795, 49796, 49797, 49798, 49799, 49800, 49
801, 49802, 49803, 49804, 49805, 49806, 49807, 49808, 49809, 49810, 49811, 49812, 49813, 49814, 49815, 49816, 49817, 4981
8, 49819, 49820, 49821, 49822, 49823, 49824, 49825, 49826, 49827, 49828, 49829, 49830, 49831, 49832, 49833, 49834, 49835,
49836, 49837, 49838, 49839, 49840, 49841, 49842, 49843, 49844, 49845, 49846, 49847, 49848, 49849, 49850, 49851, 49852, 49
853, 49854, 49855, 49856, 49857, 49858, 49859, 49860, 49861, 49862, 49863, 49864, 49865, 49866, 49867, 49868, 49869, 4987
0, 49871, 49872, 49873, 49874, 49875, 49876, 49877, 49878, 49879, 49880, 49881, 49882, 49883, 49884, 49885, 49886, 49887,
49888, 49889, 49890, 49891, 49892, 49893, 49894, 49895, 49896, 49897, 49898, 49899, 49900, 49901, 49902, 49903, 49904, 49
905, 49906, 49907, 49908, 49909, 49910, 49911, 49912, 49913, 49914, 49915, 49916, 49917, 49918, 49919, 49920, 49921, 4992
2, 49923, 49924, 49925, 49926, 49927, 49928, 49929, 49930, 49931, 49932, 49933, 49934, 49935, 49936, 49937, 49938, 49939,
49940, 49941, 49942, 49943, 49944, 49945, 49946, 49947, 49948, 49949, 49950, 49951, 49952, 49953, 49954, 49955, 49956, 49
957, 49958, 49959, 49960, 49961, 49962, 49963, 49964, 49965, 49966, 49967, 49968, 49969, 49970, 49971, 49972, 49973, 4997
4, 49975, 49976, 49977, 49978, 49979, 49980, 49981, 49982, 49983, 49984, 49985, 49986, 49987, 49988, 49989, 49990, 49991,
49992, 49993, 49994, 49995, 49996, 49997, 49998, 49999]

Execution Time in seconds: 12.225267200000005

Execution Time in seconds:      [0.012938600000005351, 0.03920820000000447, 0.07216599999999573, 0.14405919999999384, 0.
5357739999999964, 1.9435608000000002, 4.326020199999995, 7.650773699999995, 12.225267200000005]
```

```
    def quicksort_median(array, leftindex, rightindex):
        if(len(quicksort_marr)<=15):
            insertion_sort(quicksort_marr)
        elif(len(quicksort_marr)>15):
             if leftindex < rightindex:
                newpivotindex = partition_median(array, leftindex, rightindex)
                quicksort_median(array, leftindex, newpivotindex)
                quicksort_median(array, newpivotindex + 1, rightindex)

    quicksort_median(quicksort_marr, 0, len(quicksort_marr))
    print("\nQuick sort",quicksort_marr)
    end=None
    end = timeit.default_timer()
    Execution_time=end - start
    Execution_time_arr14.append(Execution_time)
    print("\nExecution Time in seconds:",Execution_time)
print("\nExecution Time in seconds:\t",Execution_time_arr14)
```

```
749, 49750, 49751, 49752, 49753, 49754, 49755, 49756, 49757, 49758, 49759, 49760, 49761, 49762, 49763, 49764, 49765, 4976
6, 49767, 49768, 49769, 49770, 49771, 49772, 49773, 49774, 49775, 49776, 49777, 49778, 49779, 49780, 49781, 49782, 49783,
49784, 49785, 49786, 49787, 49788, 49789, 49790, 49791, 49792, 49793, 49794, 49795, 49796, 49797, 49798, 49799, 49800, 49
801, 49802, 49803, 49804, 49805, 49806, 49807, 49808, 49809, 49810, 49811, 49812, 49813, 49814, 49815, 49816, 49817, 4981
8, 49819, 49820, 49821, 49822, 49823, 49824, 49825, 49826, 49827, 49828, 49829, 49830, 49831, 49832, 49833, 49834, 49835,
49836, 49837, 49838, 49839, 49840, 49841, 49842, 49843, 49844, 49845, 49846, 49847, 49848, 49849, 49850, 49851, 49852, 49
853, 49854, 49855, 49856, 49857, 49858, 49859, 49860, 49861, 49862, 49863, 49864, 49865, 49866, 49867, 49868, 49869, 4987
0, 49871, 49872, 49873, 49874, 49875, 49876, 49877, 49878, 49879, 49880, 49881, 49882, 49883, 49884, 49885, 49886, 49887,
49888, 49889, 49890, 49891, 49892, 49893, 49894, 49895, 49896, 49897, 49898, 49899, 49900, 49901, 49902, 49903, 49904, 49
905, 49906, 49907, 49908, 49909, 49910, 49911, 49912, 49913, 49914, 49915, 49916, 49917, 49918, 49919, 49920, 49921, 4992
2, 49923, 49924, 49925, 49926, 49927, 49928, 49929, 49930, 49931, 49932, 49933, 49934, 49935, 49936, 49937, 49938, 49939,
49940, 49941, 49942, 49943, 49944, 49945, 49946, 49947, 49948, 49949, 49950, 49951, 49952, 49953, 49954, 49955, 49956, 49
957, 49958, 49959, 49960, 49961, 49962, 49963, 49964, 49965, 49966, 49967, 49968, 49969, 49970, 49971, 49972, 49973, 4997
4, 49975, 49976, 49977, 49978, 49979, 49980, 49981, 49982, 49983, 49984, 49985, 49986, 49987, 49988, 49989, 49990, 49991,
49992, 49993, 49994, 49995, 49996, 49997, 49998, 49999]

Execution Time in seconds: 12.151016699999985

Execution Time in seconds:      [0.008672600000011244, 0.029781500000012784, 0.05605980000001409, 0.14926140000000032,
0.5489656000000025, 1.9946544999999958, 4.387037800000002, 7.721379400000018, 12.151016699999985]
```

```
    def quicksort_median(array, leftindex, rightindex):
        if(len(quicksort_marr)<=15):
            insertion_sort(quicksort_marr)
        elif(len(quicksort_marr)>15):
             if leftindex < rightindex:
                newpivotindex = partition_median(array, leftindex, rightindex)
                quicksort_median(array, leftindex, newpivotindex)
                quicksort_median(array, newpivotindex + 1, rightindex)

    quicksort_median(quicksort_marr, 0, len(quicksort_marr))
    print("\nQuick sort",quicksort_marr)
    end=None
    end = timeit.default_timer()
    Execution_time=end - start
    Execution_time_arr14.append(Execution_time)
    print("\nExecution Time in seconds:",Execution_time)
print("\nExecution Time in seconds:\t",Execution_time_arr14)
```

```
749, 49750, 49751, 49752, 49753, 49754, 49755, 49756, 49757, 49758, 49759, 49760, 49761, 49762, 49763, 49764, 49765, 4976
6, 49767, 49768, 49769, 49770, 49771, 49772, 49773, 49774, 49775, 49776, 49777, 49778, 49779, 49780, 49781, 49782, 49783,
49784, 49785, 49786, 49787, 49788, 49789, 49790, 49791, 49792, 49793, 49794, 49795, 49796, 49797, 49798, 49799, 49800, 49
801, 49802, 49803, 49804, 49805, 49806, 49807, 49808, 49809, 49810, 49811, 49812, 49813, 49814, 49815, 49816, 49817, 4981
8, 49819, 49820, 49821, 49822, 49823, 49824, 49825, 49826, 49827, 49828, 49829, 49830, 49831, 49832, 49833, 49834, 49835,
49836, 49837, 49838, 49839, 49840, 49841, 49842, 49843, 49844, 49845, 49846, 49847, 49848, 49849, 49850, 49851, 49852, 49
853, 49854, 49855, 49856, 49857, 49858, 49859, 49860, 49861, 49862, 49863, 49864, 49865, 49866, 49867, 49868, 49869, 4987
0, 49871, 49872, 49873, 49874, 49875, 49876, 49877, 49878, 49879, 49880, 49881, 49882, 49883, 49884, 49885, 49886, 49887,
49888, 49889, 49890, 49891, 49892, 49893, 49894, 49895, 49896, 49897, 49898, 49899, 49900, 49901, 49902, 49903, 49904, 49
905, 49906, 49907, 49908, 49909, 49910, 49911, 49912, 49913, 49914, 49915, 49916, 49917, 49918, 49919, 49920, 49921, 4992
2, 49923, 49924, 49925, 49926, 49927, 49928, 49929, 49930, 49931, 49932, 49933, 49934, 49935, 49936, 49937, 49938, 49939,
49940, 49941, 49942, 49943, 49944, 49945, 49946, 49947, 49948, 49949, 49950, 49951, 49952, 49953, 49954, 49955, 49956, 49
957, 49958, 49959, 49960, 49961, 49962, 49963, 49964, 49965, 49966, 49967, 49968, 49969, 49970, 49971, 49972, 49973, 4997
4, 49975, 49976, 49977, 49978, 49979, 49980, 49981, 49982, 49983, 49984, 49985, 49986, 49987, 49988, 49989, 49990, 49991,
49992, 49993, 49994, 49995, 49996, 49997, 49998, 49999]

Execution Time in seconds: 11.82494220000001

Execution Time in seconds:      [0.009956199999976434, 0.031337699999994584, 0.05203629999999748, 0.13562650000000076,
0.56080460000004, 2.070368500000029, 4.40967820000003, 8.19588060000001, 11.82494220000001]
```