```python
def Increase_Key(arr, i, newkey):
  if newkey < arr[i]:
    print("error: New key is smaller than current key")
  else:
    arr[i] = newkey
  while i > 0 and arr[(i - 1)//2] < arr[i]:
    arr[i], arr[(i - 1)//2] = arr[(i - 1)//2], arr[i]
    i = (i - 1)//2


def Insert(arr, key):
  arr.append(-99999)
  Increase_Key(arr, len(arr)-1, key)

"""Graphing"""
arr=[]
n=[]
max_time=[]
extract_max_time=[]
increase_key_time=[]
insert_time=[]
logn=[]
scale = 1000000

for i in range(1,1000,10):
  n.append(i)
  lg=math.log2(i)
  logn.append(lg)
  for j in range(1, i+1):
    arr.append(random.randrange(1, 10000))
  print("Initial array is: ",arr,"\n")
  buildMaxHeap(arr)

  start = timeit.default_timer()
  Maximum(arr)
  max_time.append((timeit.default_timer() - start)*scale)

  start = timeit.default_timer()
  Extract_Max(arr)
  ext_max_time = (timeit.default_timer() - start)*scale
  extract_max_time.append( ext_max_time )

  start = timeit.default_timer()
  Increase_Key(arr, 6, i)
  increase_key_time.append((timeit.default_timer() - start)*scale)

  start = timeit.default_timer()
  Insert(arr, random.randrange(1, 10000))
```

```python
"""Quick Sort Algorithm"""

import random, timeit, math, matplotlib.pyplot as plt

"""Defining Partition"""
def partition(arr, l, r):
  pivot = arr[r]
  i = l - 1
  j = 0

  # comparing all with pivot
  for j in range(l, r):
    if arr[j] <= pivot:
      i = i + 1

      # Swapping arr[i] with element at j
      (arr[i], arr[j]) = (arr[j], arr[i])

  # Swap the pivot element with the last i
  (arr[i + 1], arr[r]) = (arr[r], arr[i + 1])

  # Return the position of pivot
  return i + 1


"""Defining Quicksort"""
def quick_sort(arr, l, r):
  if l < r:
    pivot = partition(arr, l, r)
    #left partition
    quick_sort(arr, l, pivot - 1)
    #right partition
    quick_sort(arr, pivot + 1, r)


"""Defining Quicksort driver"""
# The main function to sort an array of given size
def quickSort(arr):
      quick_sort(arr, 0, len(arr)-1)


"""Function to return array of random numbers of required sizes"""
def rand_arr(n):
  return [random.randrange(100) for i in range(n)]

print("\n\tGenerating Random arrays of different sizes...")
for i in range(5,8):
  print("random array of size",i,":",rand_arr(i))
```

```python
def curr_time(): return timeit.default_timer()*scale

"""testing QuickSort"""
print("\n\tRunning QuickSort on few examples")
for i in range(3):
       array = rand_arr(10)
       print("original array : ", array)
       quickSort(array)
       print("sorted array   : ", array ,'\n')


print("\n\tRunning algorithm for large values ... ")
"""Graphing"""
arr=[]
n=[]
qtime=[]
nlogn=[]
scale = 10000

for i in range(1,1000,5):
  n.append(i)
  nlogn.append(i*math.log2(i))
  for j in range(1, i+1):
    arr.append(random.randrange(1, 10000))
  print("Initial array is: ",arr,"\n")

  start = curr_time()

  quickSort(arr)

  end = curr_time()
  total_time = (end - start)
  qtime.append( total_time )
  print("Time for execution, n = {}: {}".format(i, total_time ))


"""Plotting this data with nlogn"""
print("\n\tPlotting graph of running time ... ")
# plotting nlogn
plt.title("nlogn vs QuickSort graph")
plt.xlabel("Input Size")
plt.ylabel("Running Times")

# plotting quicksort graph
plt.plot(n, qtime, color="blue")

#plotting nlogn graph
plt.plot(n, nlogn, color="red")
```

```
plt.gca().legend(('QuickSort','nlogn'))

#showing the combined plot
plt.show()

print("finished ... ")
```

Plot: