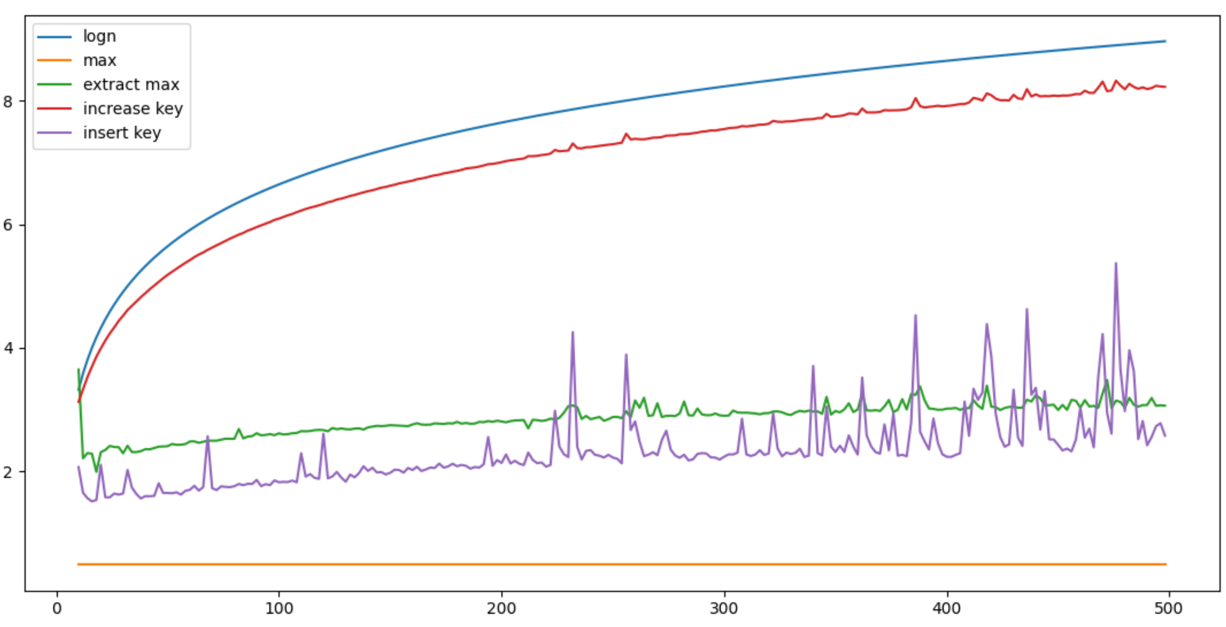
```python
"""Heap Sort Algorithm"""

import random, timeit, numpy as np, matplotlib.pyplot as plt

"""Defining Buildheap"""
def buildMaxHeap(arr):
  n = len(arr)
  # Since last parent will be at ((n//2)-1) we can start from there till 0.
  for i in range(n//2 - 1, -1, -1):
    heapify(arr, n, i)


"""Defining Heapify"""
def heapify(arr, n, i):
  largest = i
  l = 2 * i + 1    # left  = 2*i + 1
  r = 2 * i + 2    # right = 2*i + 2
  if l < n and arr[i] < arr[l]:
    largest = l
  if r < n and arr[largest] < arr[r]:
    largest = r
  if largest != i:
    arr[i],arr[largest] = arr[largest],arr[i] # swap
    heapify(arr, n, largest)


"""Defining Heapsort"""
# The main function to sort an array of given size
def heapsort(arr):
  n = len(arr)
  #build a maxheap
  buildMaxHeap(arr)
  # One by one extract elements and swap with root
  for i in range(n-1, 0, -1):
    arr[i], arr[0] = arr[0], arr[i] # swap
    heapify(arr, i, 0)


"""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))

def curr_time(): return timeit.default_timer()*10000
```

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


"""Function to note time for execution"""
def running_times(function, a, b, h):
  times = []
  for i in range(a,b,h):
    start = curr_time()
    function( rand_arr(i) )
    end   = curr_time()
    total_time = (end - start)
    times.append(total_time)
    print("Time for execution, n = {}: {}".format(i, total_time))
  return times


"""Running Heapsort for large size arrays"""
print("\n\tRunning algorithm for large values... ")
a = 1
b = 10000
h = 20

# data to be plotted
x = np.arange(a, b, h)
y = x * np.log2(x)
z = running_times(heapsort, a,b,h)
print("ran for {} values from {} to {}".format((b-a)//h, a, b))
"""Plotting this data with nlogn"""
print("\n\tPlotting graph of running time... ")
# plotting nlogn
plt.title("nlogn vs heapsort graph")
plt.xlabel("Input Size")
plt.ylabel("Running Times")

# plotting mergesort graph
plt.plot(x, z, color="blue")

#plotting nlogn graph
plt.plot(x, y, color="red")

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

```
"""
# Implementing Priority Queue
"""

import random, timeit, sys, math
import numpy as np
import matplotlib.pyplot as plt

"""Defining Buildheap"""
def buildMaxHeap(arr):
  n = len(arr)
  for i in range(n//2 - 1, -1, -1):
    heapify(arr, n, i)

"""Defining Heapify"""
def heapify(arr, n, i):
  largest = i
  l = 2 * i + 1 # left  = 2*i + 1
  r = 2 * i + 2 # right = 2*i + 2

  #if left child of root exists and is greater than root
  if l < n and arr[i] < arr[l]:
    largest = l

  #if right child of root exists and is greater than root
  if r < n and arr[largest] < arr[r]:
    largest = r

  if largest != i:
    arr[i],arr[largest] = arr[largest],arr[i] # swap
    heapify(arr, n, largest)

"""Defining the Operations"""
def Maximum(arr):
  return arr[0]


def Extract_Max(arr):
  n = len(arr)
  if n < 1:
    print("error: heap underflow")
    return
  max = arr[0]
  arr[0] = arr[n-1]
  arr.pop()
  heapify(arr, 0, n)
  return max
```

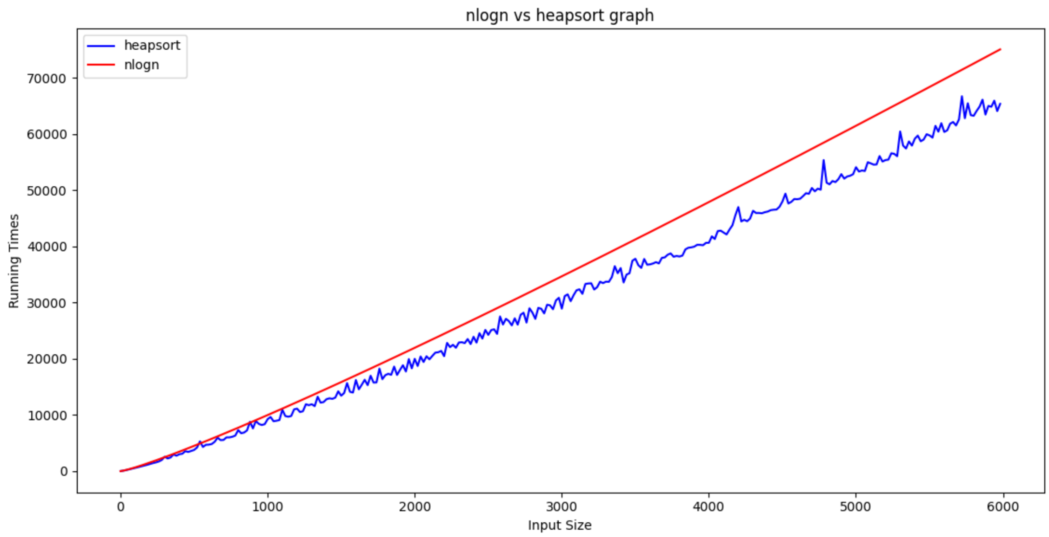
```
def Increase_Key(arr, i, newkey):
  if newkey < arr[i]:
    1+1
    #print("error: New key is smaller than current key")
  else:
    arr[i] = newkey
  while i > 0 and arr[(i - 1)//2] < arr[i]:
    arr[i], arr[(i - 1)//2] = arr[(i - 1)//2], arr[i]
    i = (i - 1)//2


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

"""Graphing"""
n=[]
arr=[]
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, i)
  insert_time.append((timeit.default_timer() - start)*scale)

plt.plot(n,logn,label="logn")
plt.plot(n,max_time,label="max")
plt.plot(n,extract_max_time,label="extract max")
plt.plot(n,increase_key_time,label="increase key")
plt.plot(n,insert_time,label="insert key")
plt.legend()
plt.show()

print("finished…")
```

Plot :



```
#showing the combined plot
plt.show()

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

Plot :


nlogn vs heapsort graph
heapsort
nlogn
Running Times
Input Size