***Group 1 Algorithm Project***

Contributors: ***Safiu Waheed (206454)***

***Kayode Peter (208077)***

***Chinedu Promise (213930)***

Department: ***Computer Science***

Algorithms: ***Insertion Sort***

***Quick Sort***

***Heap Sort***

**Combinational Table Of Insertion Sort, Quick Sort And Heap Sort Algorithms**

**Table 1.1 Parameters for Data Sizes and Time Taken**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **S/N** | **Data Sizes** | **Insertion sort** | **Quick sort** | **Heap sort** |
| 1 | 10, 000 | 35.2065 | 0.2226 | 0.539 |
| 2 | 20, 000 | 135.2269 | 0.4257 | 1.1757 |
| 3 | 30, 000 | 305.4880 | 0.7968 | 1.8515 |
| 4 | 40, 000 | 543.8278 | 0.953 | 2.5936 |
| 5 | 50, 000 | 973.6471 | 1.2499 | 3.3045 |

**Figure 1.1 Graph of Data Size against Time Taken**

**Insertion Sort**

**Code**

#Implementation of Insertion sorting

import time

start = time.time()

import random

arr = []

for i in range(0,50000):

n = random.randint(0, 50000)

arr.append(n)

print(arr)

def insertion\_sort(arr):

for i in range(1,len(arr)):

j = i

while arr[j-1] > arr[j] and j > 0:

arr[j-1], arr[j] = arr[j], arr[j-1]

j -= 1

insertion\_sort(arr)

print("\nThe sorted array is:")

print(arr)

end = time.time()

print ("\nThe Runtime of the program is ",(end - start))

**Table 1.2 Parameters for Data Sizes and Time Taken**

|  |  |  |
| --- | --- | --- |
| **S/N** | **Data Sizes** | **Time taken(seconds)** |
| 1 | 10, 000 | 35.2065 |
| 2 | 20, 000 | 135.2269 |
| 3 | 30, 000 | 305.4880 |
| 4 | 40, 000 | 543.8278 |
| 5 | 50, 000 | 973.6471 |

**Figure 1.2 Graph of Insertion Sort (Time Taken against Data Size)**

**Quick sort**

**Code**

#Implementation of Quick sort

import time

start = time.time()

import random

arr = []

for i in range(0,50000):

n = random.randint(0, 50000)

arr.append(n)

print(arr)

def quicksort(arr, left, right):

if left < right:

partition\_pos = partition(arr, left, right)

quicksort(arr, left, partition\_pos-1)

quicksort(arr, partition\_pos+1, right)

def partition(arr, left, right):

i = left

j = right-1

pivot = arr[right]

while i < j:

while i < right and arr[i] < pivot:

i += 1

while j > left and arr[j] >= pivot:

j -= 1

if i < j:

arr[i], arr[j] = arr[j], arr[i]

if arr[i] > pivot:

arr[i], arr[right] = arr[right], arr[i]

return i

quicksort(arr, 0, len(arr)-1)

print("\nThe sorted array is:")

print(arr)

end = time.time()

print ("\nThe Runtime of the program is ",(end - start))

**Table 1.3 Parameters for Data Sizes and Time Taken**

|  |  |  |
| --- | --- | --- |
| **S/N** | **Data Sizes** | **Time taken(seconds)** |
| 1 | 10, 000 | 0.2226 |
| 2 | 20, 000 | 0.4257 |
| 3 | 30, 000 | 0.7968 |
| 4 | 40, 000 | 0.9530 |
| 5 | 50, 000 | 1.2499 |

**Figure 1.3 Graph of Quick Sort (Time Taken against Data Size)**

**Heap Sort**

**Code**

#Implementation of Heap Sort

import time

start = time.time()

import random

arr = []

for i in range(0, 50000):

n = random.randint(0, 50000)

arr.append(n)

print(arr)

def swap(arr, i, j):

arr[i], arr[j] = arr[j], arr[i]

def siftDown(arr, i, upper):

while (True):

l, r = i\*2+1, i\*2+2

if max(l, r) < upper:

if arr[i] >= max(arr[l], arr[r]): break

elif arr[l] > arr[r]:

swap(arr, i, l)

i = l

else:

swap(arr, i, r)

i = r

elif l < upper:

if arr[l] > arr[i]:

swap(arr, i, l)

i = l

else: break

elif r < upper:

if arr[r] > arr[i]:

swap(arr, i, r)

i = r

else: break

else: break

def heapsort(arr):

for j in range((len(arr)-2)//2, -1, -1):

siftDown(arr, j, len(arr))

for end in range(len(arr)-1, 0, -1):

swap(arr, 0, end)

siftDown(arr, 0, end)

#arr = [5, 16, 8, 14, 20, 1, 26]

heapsort(arr)

print("\nThe sorted array is:")

print(arr)

end = time.time()

print ("\nThe Runtime of the program is ",(end - start))

**Table 1.4 Parameters for Data Sizes and Time Taken**

|  |  |  |
| --- | --- | --- |
| **S/N** | **Data Sizes** | **Time taken(seconds)** |
| 1 | 10, 000 | 0.5390 |
| 2 | 20, 000 | 1.1757 |
| 3 | 30, 000 | 1.8515 |
| 4 | 40, 000 | 2.5936 |
| 5 | 50, 000 | 3.3045 |

**Figure 1.4 Graph of Heap Sort (Time Taken against Data Size)**

**Observations, Inferences and Conclusions**

According to **table 1.1** we observed that with every increase in the data size leads to an increase in the time taken to sort. We can also observe that the quick sort algorithm takes less time to sort large data sets compared to the heap and insertion sort. We also observe that the insertion sort algorithm takes more time to sort large data sets .From the analysis of **figure 1.1** we can conclude that for large data sets the most efficient and fastest sorting algorithm out of the three (insertion, heap and quick sort) is the **quick sort** algorithm and the worst case algorithm is the **insertion algorithm**.