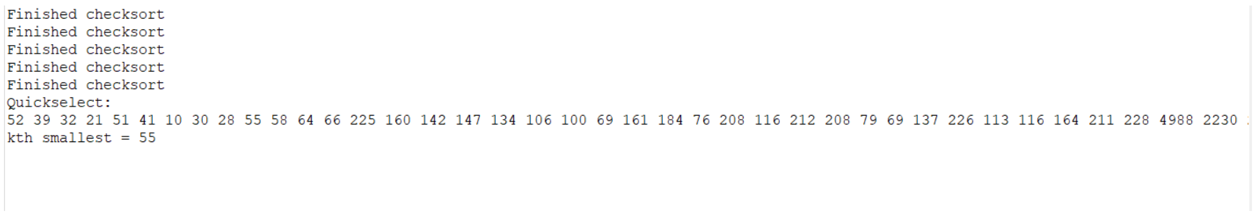
I confirm that I will keep the content of this assignment confidential. I confirm that I have not received any unauthorized assistance in preparing for or writing this assignment. I acknowledge that a mark of 0 may be assigned for copied work." + Arun Reddy Nalla + 110088379

Task 1: Use class Sort.java provided in class, the dual-pivot Quicksort of Java 8 (Arrays.sort), and RadixSort.java provided in class.

Solution:
These are class which are already provided in resources.

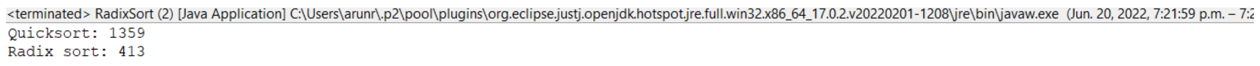
File name:
Sort.java and RadixSort.java

Output:



```
Finished checksort
Finished checksort
Finished checksort
Finished checksort
Finished checksort
Quickselect:
52 39 32 21 51 41 10 30 28 55 58 64 66 225 160 142 147 134 106 100 69 161 184 76 208 116 212 208 79 69 137 226 113 116 164 211 228 4988 2230
kth smallest = 55
```

Output for Sort.java



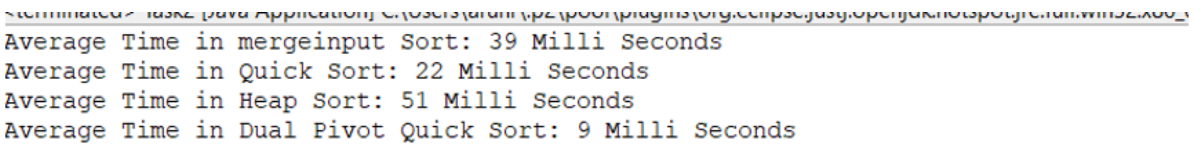
```
<terminated> RadixSort (2) [Java Application] C:\Users\arunr\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.2.v20220201-1208\jre\bin\javaw.exe  (Jun. 20, 2022, 7:21:59 p.m. – 7:2
Quicksort: 1359
Radix sort: 413
```

Output for RadixSort.java

Task 2 :

Do the following for Mergesort, Quicksort, Heapsort and dual-pivot Quicksort:
a. Create 100,000 random keys (of type long) and sort them. Repeat this 100 times.
b. Compute the average CPU time taken to sort the keys for the four methods.
c. Comment on the results and compare them to the average-case complexities discussed in class.

Soultion :



```
Average Time in mergeinput Sort: 39 Milli Seconds
Average Time in Quick Sort: 22 Milli Seconds
Average Time in Heap Sort: 51 Milli Seconds
Average Time in Dual Pivot Quick Sort: 9 Milli Seconds
```

Output for Task2.java

Note: Time is measured in Milli second

I have created a class Task2 for this solution. In this program I have used Sort.java and RadixSort.java given in the resources for sorting the key in different method.

First, I had created 100,000 random keys and stored in arrays. Then with help Sort.java and RadixSort.java I had sorted these arrays and calculated total time taken for all type of sorting method given the question.

Then I had repeated this process for 100 times and then calculated average time taken for sorting in those four types sorting.

You can find the results in output for Task2.java image.

2.c) As, discussed in the class chapter 4 sorting

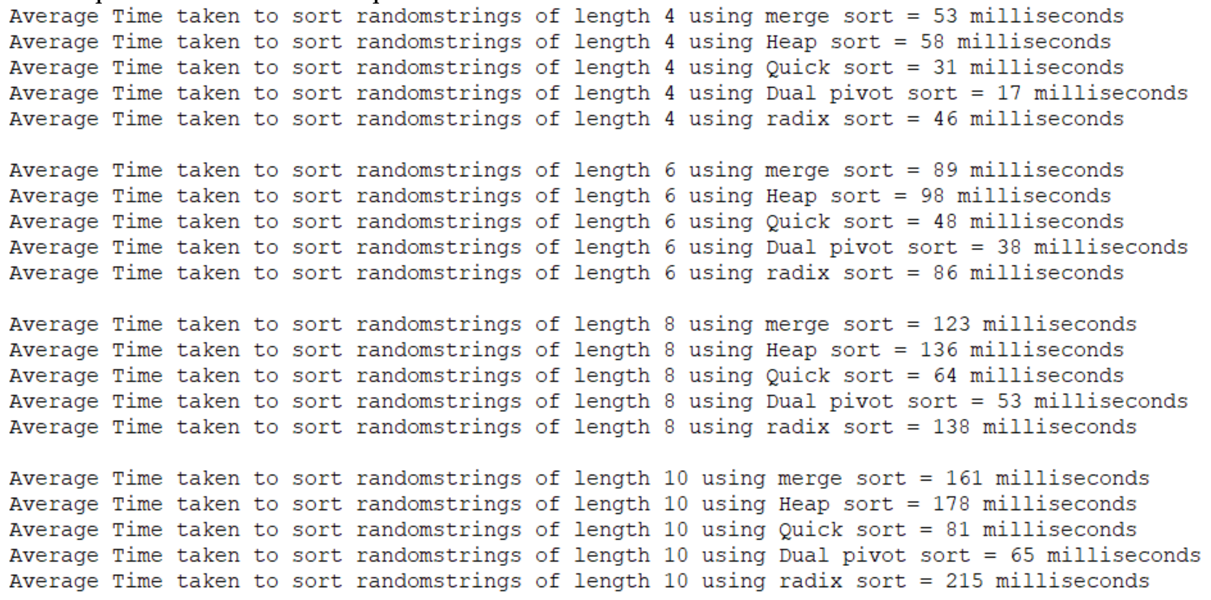
- The average time complexity for all Mergesort, Quicksort, Heapsort and dual-pivot Quicksort is O(n logn)
- For radix sort average time complexity is O(d(n + N))
    - If d = log n and N = O(n), Radix sort runs in O(n log n)
    - If d is constant and N = O(n), Radix sort runs in O(n)

- From my observation, Dual pivot Quick sort is taking least amount of time for sorting 100,000 random keys followed by Quick sort and merge sort. Heap sort is taking more time to sort 100,000 keys

Task 3 :
Do the following for the four sorting methods of #2, and for Radix sort:
a. Create 100,000 random strings of length 4 and sort them using the five sorting methods.
b. Repeat (a) 10 times and compute the average CPU time that takes to sort the keys for the five methods.
c. Repeat (a) and (b) with strings of length 6, 8, 10.
d. Create a table with the results and compare the times with the average-case and worstcase complexities as studied in class.

For this activity, I created the Task3 class in package "assignment" in the Assignment2 JAVA Project.

This program generates random strings of lengths 4, 6, 8, and 10 and sorted them using the techniques described in the question.



```
Average Time taken to sort randomstrings of length 4 using merge sort = 53 milliseconds
Average Time taken to sort randomstrings of length 4 using Heap sort = 58 milliseconds
Average Time taken to sort randomstrings of length 4 using Quick sort = 31 milliseconds
Average Time taken to sort randomstrings of length 4 using Dual pivot sort = 17 milliseconds
Average Time taken to sort randomstrings of length 4 using radix sort = 46 milliseconds

Average Time taken to sort randomstrings of length 6 using merge sort = 89 milliseconds
Average Time taken to sort randomstrings of length 6 using Heap sort = 98 milliseconds
Average Time taken to sort randomstrings of length 6 using Quick sort = 48 milliseconds
Average Time taken to sort randomstrings of length 6 using Dual pivot sort = 38 milliseconds
Average Time taken to sort randomstrings of length 6 using radix sort = 86 milliseconds

Average Time taken to sort randomstrings of length 8 using merge sort = 123 milliseconds
Average Time taken to sort randomstrings of length 8 using Heap sort = 136 milliseconds
Average Time taken to sort randomstrings of length 8 using Quick sort = 64 milliseconds
Average Time taken to sort randomstrings of length 8 using Dual pivot sort = 53 milliseconds
Average Time taken to sort randomstrings of length 8 using radix sort = 138 milliseconds

Average Time taken to sort randomstrings of length 10 using merge sort = 161 milliseconds
Average Time taken to sort randomstrings of length 10 using Heap sort = 178 milliseconds
Average Time taken to sort randomstrings of length 10 using Quick sort = 81 milliseconds
Average Time taken to sort randomstrings of length 10 using Dual pivot sort = 65 milliseconds
Average Time taken to sort randomstrings of length 10 using radix sort = 215 milliseconds
```

Output for Task3.java

I have created a class Task3 for this solution. In this program I have used Sort.java and RadixSort.java given in the resources for sorting the key in different method.

First, I had created 100,000 random strings of length 4 and stored in arrays. Then with help Sort.java and RadixSort.java I had sorted these arrays and calculated total time taken for all type of sorting method given the question.

Then I had repeated this process for 10 times with different four different type of length {4,6,8,10} and then calculated average time taken for sorting in those four types sorting.

You can find the results in output for Task3.java image.

Table representation for Task3.java output

String Length	4	6	8	10
Merge sort	53	89	123	161
Heap sort	58	98	136	178
Quick sort	31	48	64	81
Dual pivot sort	17	38	53	65
Radix sort	46	86	138	215

Note : Time is measured in milli seconds.

Task 4
Comment on: which sorting method will you use in your applications? in which case? Why?

In general, quick sort takes less time for sorting. So quick sort will used frequently.
But we could not decide quick sort is best based on one situation. its varies depends on real life situations.

Merge sort

Merge sort is based on Divide and conquer algorithm. This is a good choice when you need an efficient sort or when you need to sort a large data set. It is stable, but it takes up a lot of space and is difficult to implement. Merge sort can also be used to sort linked lists.

Quick sort

Quick sort is also based on Divide and conquer algorithm. This is the fastest only but not stable. This is a good if dataset is small and fits in the memory, then you can get efficient and fast sorting when compared to merge sort

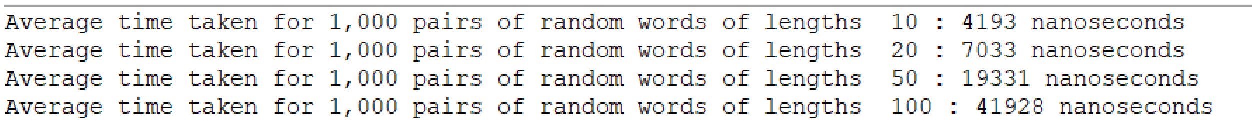
Heap sort

In heap sort, sorting is made by creating of heaps (min heap or max heap). So, this type of sorting is used to find the smallest or largest number quickly.

Radix sort

Radix sort is used in parallel computing applications since it sort all the digits from least significant digit to the most significant digit.

Task 5
5. Use the edit distance (class Sequences.java) implementation provided in the source code.
a. Generate 1,000 pairs of random words of lengths 10, 20, 50 and 100.
b. Compute the edit distance for all words and find the average CPU time for each pair.
c. Compare the CPU times obtained for each word length with the running times of the edit distance algorithm.



```
Average time taken for 1,000 pairs of random words of lengths  10 : 4193 nanoseconds
Average time taken for 1,000 pairs of random words of lengths  20 : 7033 nanoseconds
Average time taken for 1,000 pairs of random words of lengths  50 : 19331 nanoseconds
Average time taken for 1,000 pairs of random words of lengths  100 : 41928 nanoseconds
```

Output for Task5.java

For this solution, I have used Sequeces.java provided in resources. In this program, I have a created a method genarateRandomword() to created a word of given length.

Then I used editDistance() to find distance between two words generated by genarateRandomword() method. And also find the CPU time taken to this process.

I had repeated this process for 1000 pairs of random words of length 10, 20, 50 and 100.
And also found average time for each length.

You can find results in output for Task5.java image.

Algorithm EditDistance runs in O(nm) as discussed in the class chapter 5 Advanced Design and Analysis.
n and m are length of two words compared.

Also, from results the time to calculate distance between words increase with increase in length of the words. So, we can say the average CPU time increases, with increase in the word length.