Fulfill numerical experiments of Example 4 presented in [1]. For this purpose, use any sorting method, determine its time complexity, code ranking/sorting of web search results (pages) and choos ing the 10 most important w ithin 34,279 results. Use any programming language. Describe the problem statement, input data, method, code and results obtained. Compare obtained results w ith ones described in Example 4 of [1]. Upload the file cons isting of the code and related report to the UNT Canvas environment.

In [7]:
```
import random

values = random.sample(range(1, 34279), 100)
print(values)
```

```
[30947, 17593, 15101, 912, 8971, 17236, 18474, 31535, 22328, 12969, 8702, 25238, 2490, 12361, 27801, 1
7704, 17959, 5908, 941, 13337, 22091, 202, 25383, 22311, 12681, 32216, 6270, 15224, 32801, 31908, 1724
2, 22580, 1268, 29611, 20472, 22070, 2834, 32579, 27431, 24000, 16295, 19988, 27571, 19349, 25737, 741
6, 7258, 28980, 3363, 1013, 56, 27110, 18228, 9193, 26606, 9886, 11467, 31708, 16411, 24959, 1458, 334
70, 9582, 19421, 6855, 17392, 17950, 33939, 23429, 1325, 28823, 3613, 14435, 15222, 17929, 10821, 228
1, 13712, 15170, 8512, 13911, 12913, 17303, 14705, 9152, 12218, 13705, 19815, 31072, 12510, 32357, 644
9, 5292, 769, 5436, 446, 11569, 1418, 20842, 5215]
```

In [8]:
```
# Quick sort in Python
# time complexity O(n log n)

def QuickSort(arr):

    elements = len(arr)

    if elements < 2:
        return arr

    current_position = 0

    for i in range(1, elements):
         if arr[i] <= arr[0]:
              current_position += 1
              temp = arr[i]
              arr[i] = arr[current_position]
              arr[current_position] = temp

    temp = arr[0]
    arr[0] = arr[current_position]
    arr[current_position] = temp

    left = QuickSort(arr[0:current_position])
    right = QuickSort(arr[current_position+1:elements])

    arr = left + [arr[current_position]] + right

    return arr

array_to_be_sorted = values
print("Original Array: ",array_to_be_sorted)
print("\n")
print("Sorted Array of top 10 results: ",QuickSort(array_to_be_sorted[:10]))
```

```
Original Array:  [30947, 17593, 15101, 912, 8971, 17236, 18474, 31535, 22328, 12969, 8702, 25238, 249
0, 12361, 27801, 17704, 17959, 5908, 941, 13337, 22091, 202, 25383, 22311, 12681, 32216, 6270, 15224,
32801, 31908, 17242, 22580, 1268, 29611, 20472, 22070, 2834, 32579, 27431, 24000, 16295, 19988, 27571,
19349, 25737, 7416, 7258, 28980, 3363, 1013, 56, 27110, 18228, 9193, 26606, 9886, 11467, 31708, 16411,
24959, 1458, 33470, 9582, 19421, 6855, 17392, 17950, 33939, 23429, 1325, 28823, 3613, 14435, 15222, 17
929, 10821, 2281, 13712, 15170, 8512, 13911, 12913, 17303, 14705, 9152, 12218, 13705, 19815, 31072, 12
510, 32357, 6449, 5292, 769, 5436, 446, 11569, 1418, 20842, 5215]
```

```
Sorted Array of top 10 results:  [912, 8971, 12969, 15101, 17236, 17593, 18474, 22328, 30947, 31535]
```

Description:

For sorting a large number that is "34279", I have picked quick sort. It has time complexity O(n log n).

Like Merge Sort, QuickSort is a Divide and Conquer algorithm. It picks an element as a pivot and partitions the given array around the picked pivot. There are many different versions of quickSort that pick pivot in different ways.

Always pick the first element as a pivot. Always pick the last element as a pivot (implemented below) Pick a random element as a pivot. Pick median as the pivot.

The key process in quickSort is a partition(). The target of partitions is, given an array and an element x of an array as the pivot, put x at its correct position in a sorted array and put all smaller elements (smaller than x) before x, and put all greater elements (greater than x) after x. All this should be done in linear time.

Problem Statement: Sort the numbers that are random in range between 1 to 34,279 and print top 10 results.

Input data: I have taken an array of 100 numbers that are randomly generated within the given range.

Method: As described above, that is Quick sort algorithm.

Code: It is same as above.

Results: Printed the top 10 sorted elements or numbers from the sorted array.

In [ ]: