

ITCS 6114/8114 – *Algorithm and Data Structures*

Project Part - 1

- Submitted by: *TARUN CHUNCHU*
(800888999)

1. Merge Sort:

Data structures chosen: Array of N elements which are randomly generated using random function.

Methods involved:

- Merge (left, right)
- Merge_sort (A)

Complexity Analysis:

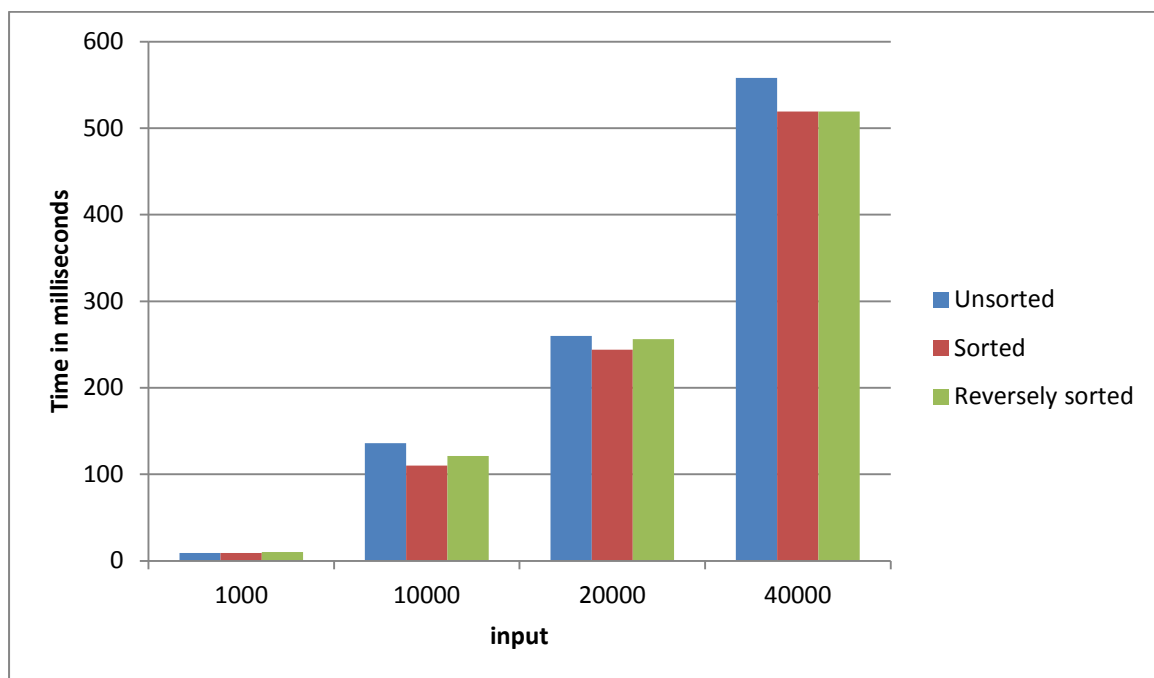
Merge sort: This method runs in $O(n \log n)$ time. It is used to sort an array in place.

Best case: $O(n \log(n))$

Average case: $O(n \log(n))$

Worst case: $O(n \log(n))$

Graph for merge sort showing execution time for 3 cases:



2.Heap Sort:

Data structures chosen: Array of N elements which are randomly generated using random function.

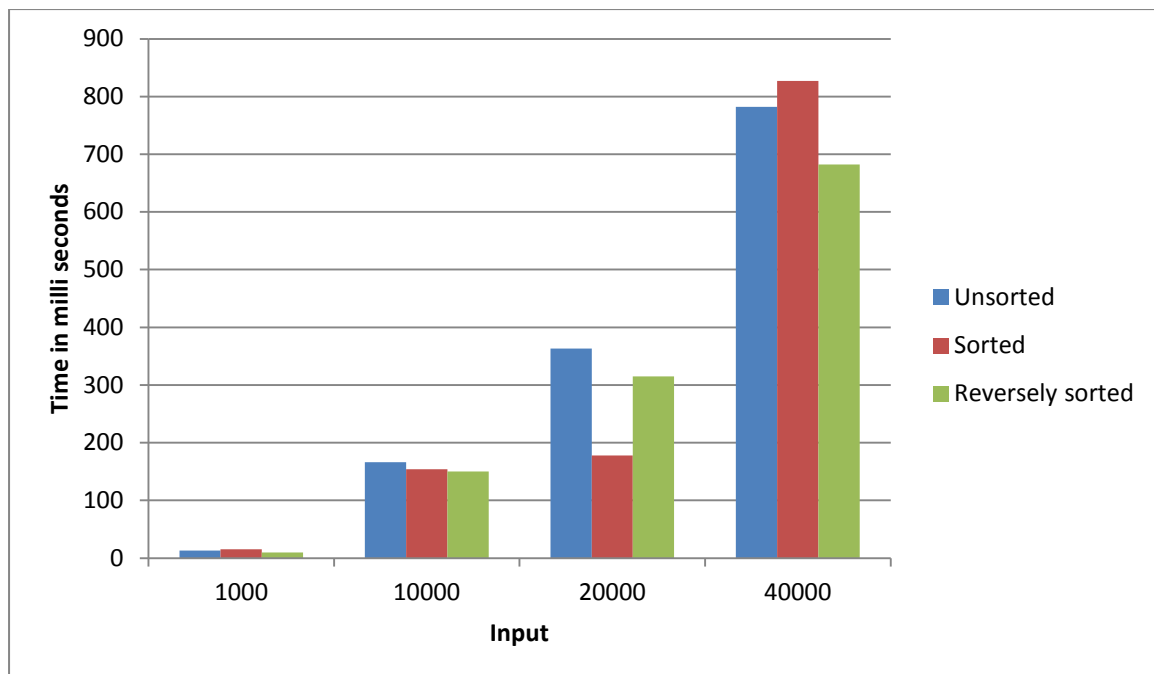
Complexity analysis:

Best case: $O(n\log(n))$

Average case: $O(n\log(n))$

Worst case: $O(n\log(n))$

Graph showing execution times of heap sort in 3 cases:



3. In – Place Quick Sort:

Data structures chosen: Array of N elements which are randomly generated using random function. Here Pivot is chosen randomly.

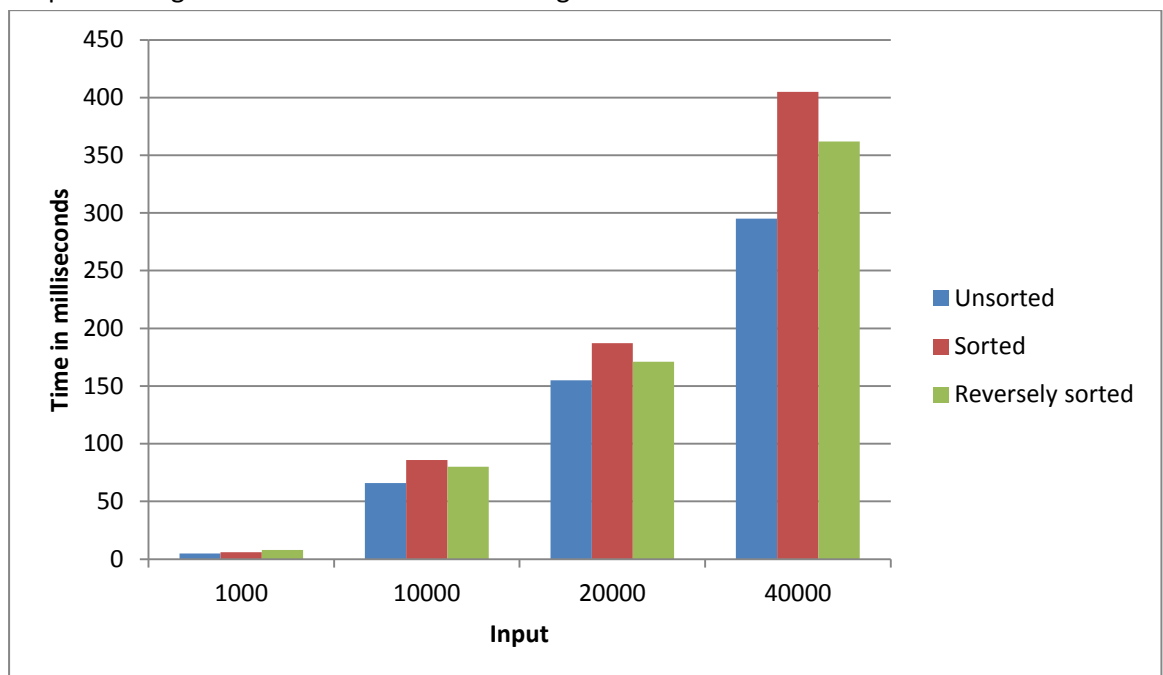
Complexity analysis:

Best case: $O(n\log(n))$

Average case: $O(n\log(n))$

Worst case: $O(n^2)$

Graph showing execution time of Quick sort algorithm in 3 cases:



4. In – Place Quick Sort – Median of Three and Insertion sort:

Data structures chosen: Array of N elements which are randomly generated using random function.

In this method we take first , middle and last element of the array and we compare among them then we move median to the center and then we move it side of first position.

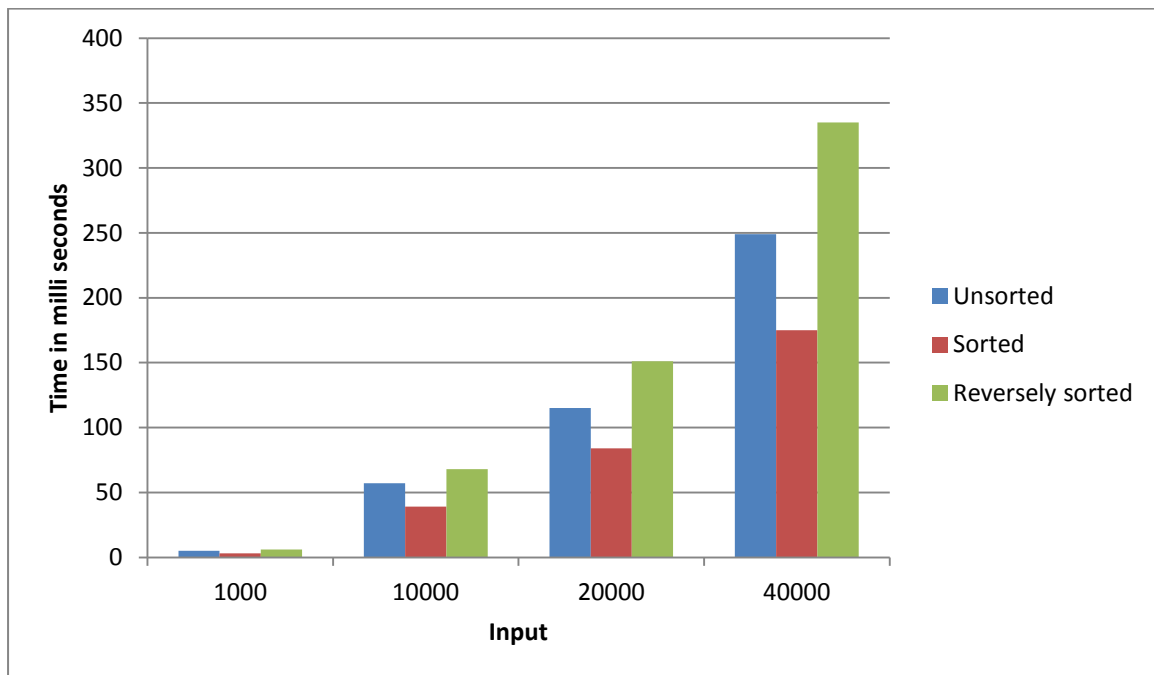
Complexity analysis:

Best case: $O(n\log(n))$

Average case: $O(n\log(n))$

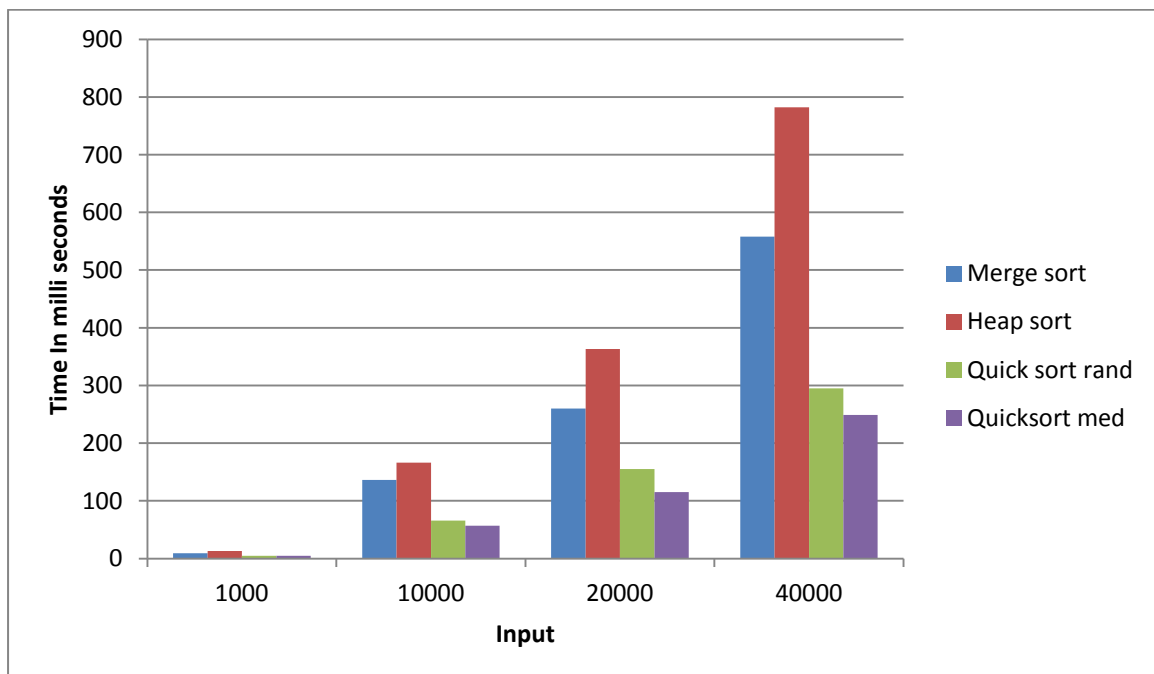
Worst case: $O(n^2)$

Graph showing execution time of Quick sort- Median as pivot in 3 cases:

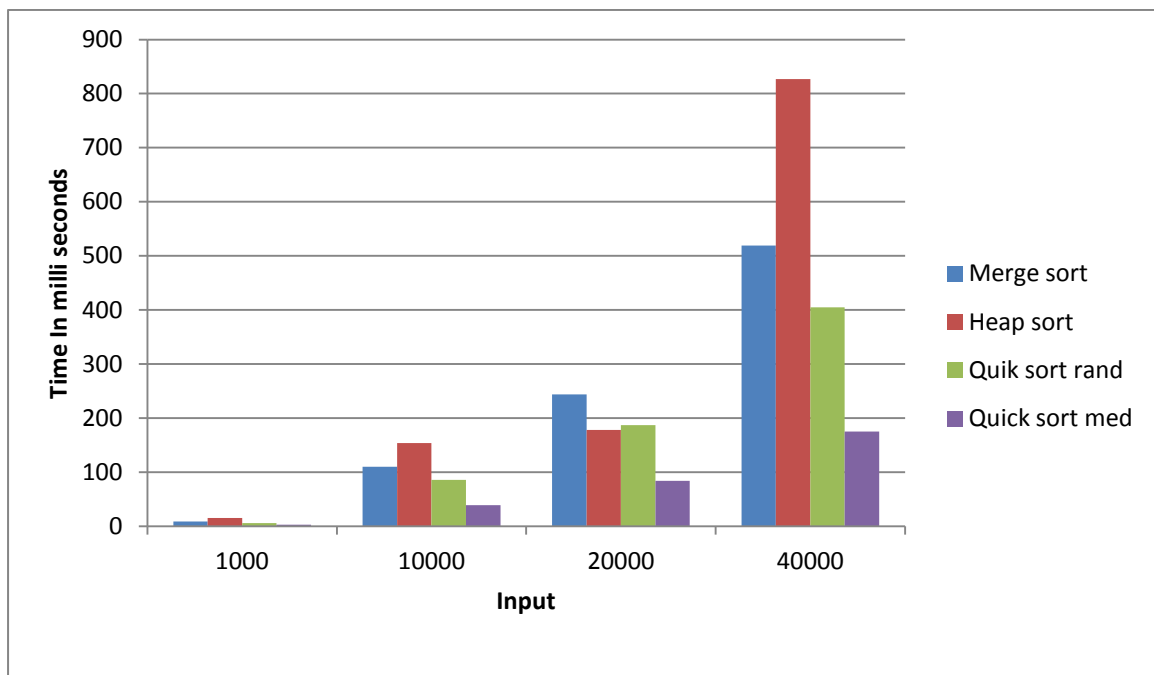


Here are graphs which compares algorithms in three different scenarios such as unsorted, sorted and reversely sorted.

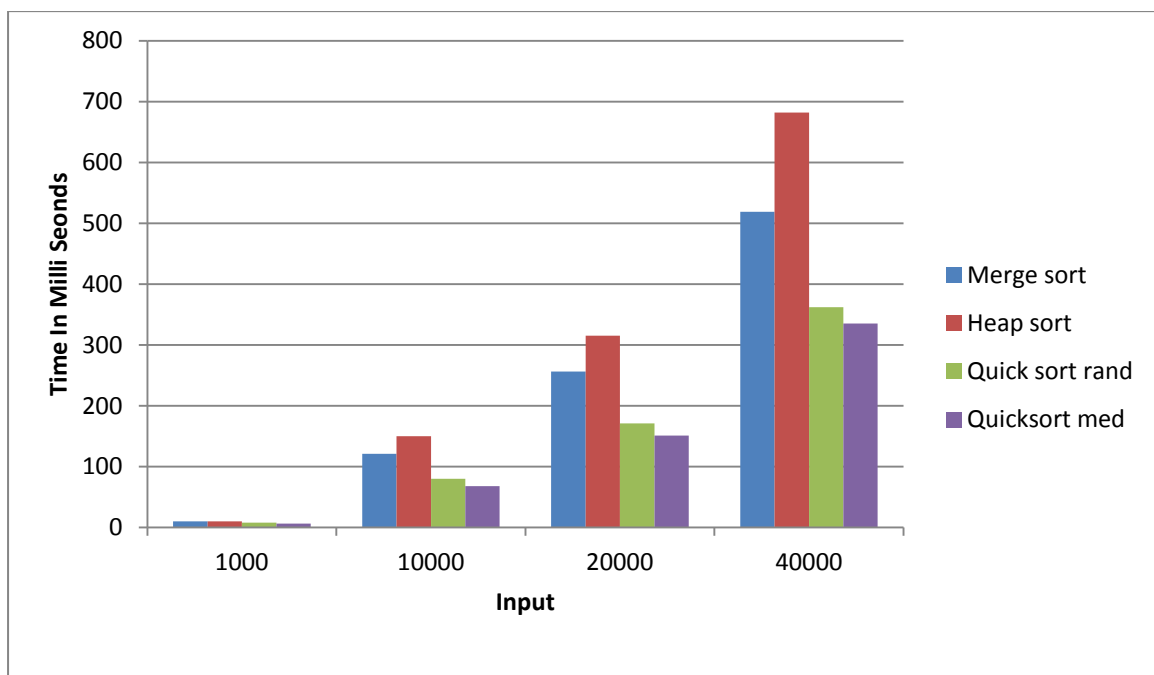
Graph showing comparison of algorithms for unsorted input :



Graph showing execution time of sorted input:



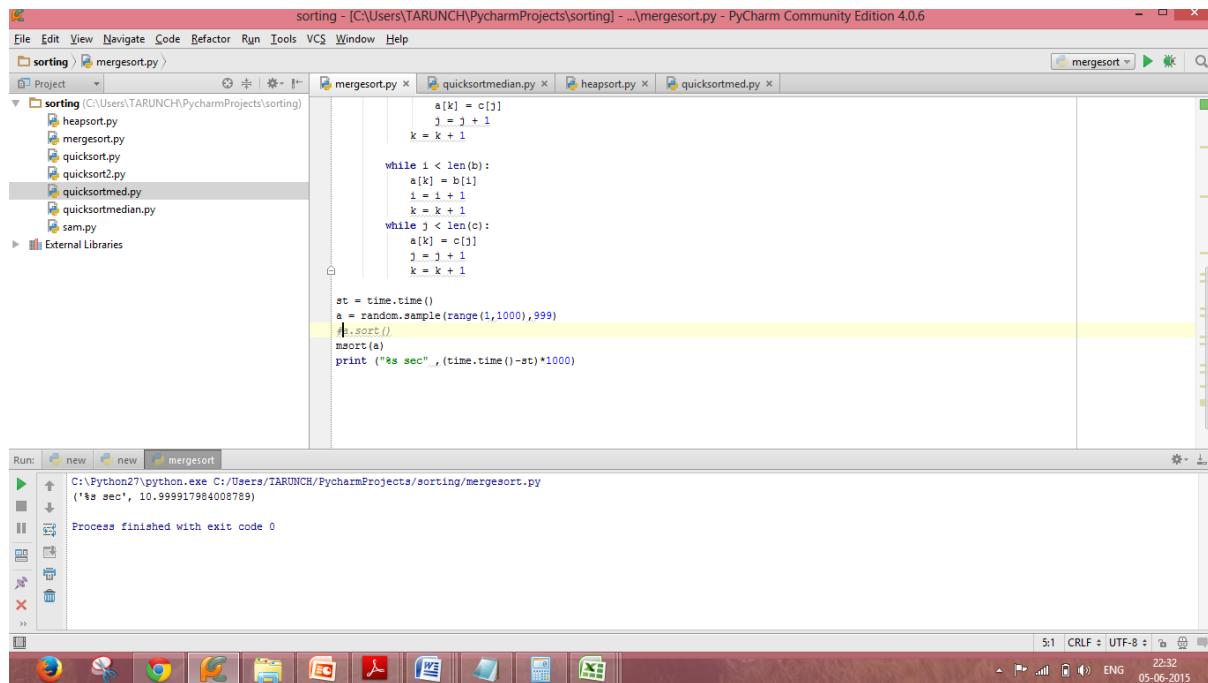
Graph showing execution time for reversely sorted input:



Analysis:

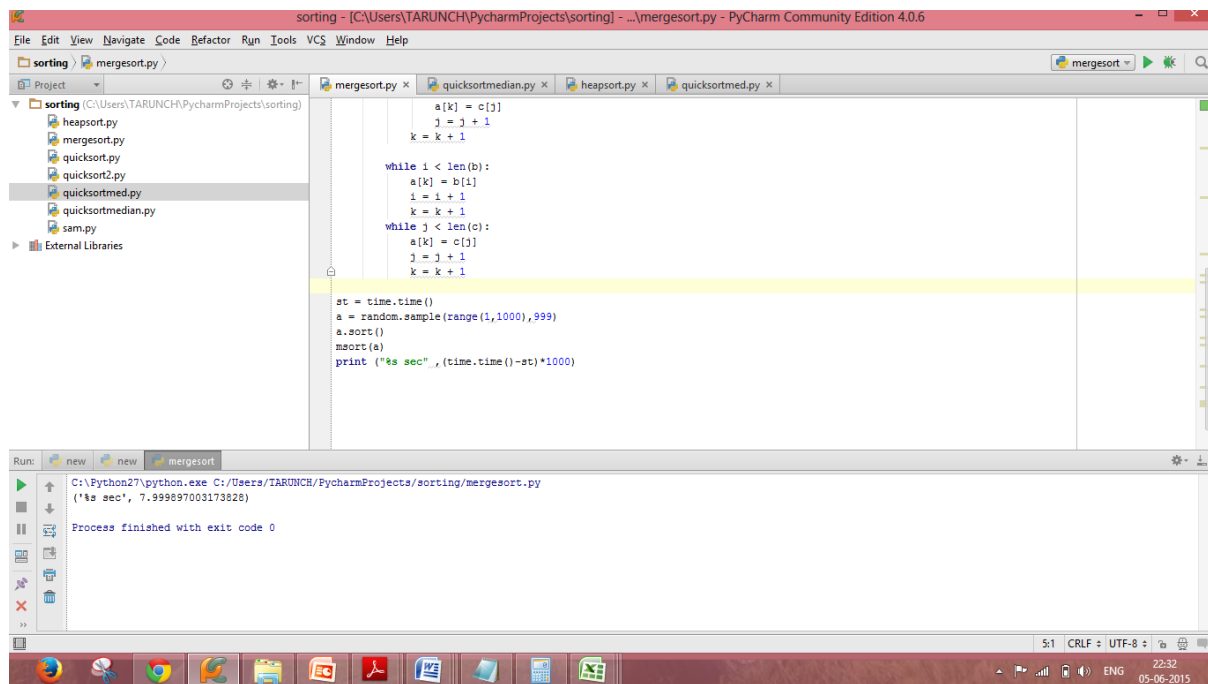
- Upon observing the entire cases heap sorting algorithm consuming more time than merge sort as it takes time to heapify. Big oh notation for both the algorithms is same $n\log(n)$ but while coming to implementation heap sort it taking more time than merge algorithm.
- Quick sort with pivot as median executed faster than quick sort in which pivot chosen randomly.
- In inplace quick sort(pivot chosen randomly) sorted array takes more time than reversely sorted array.
- Execution time of heap sort increases very much corresponding to increase of input size.

Merge sort execution: (Unsorted case)



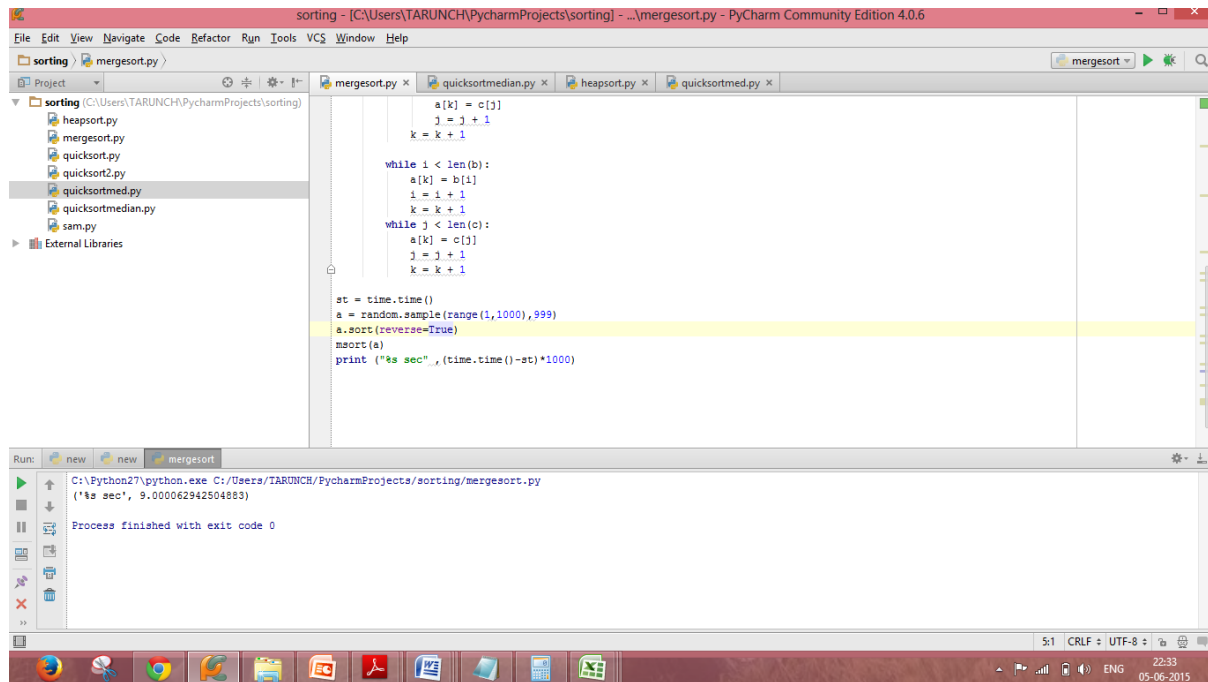
```
sorting - [C:\Users\TARUNCH\PycharmProjects\sorting] - ... \mergesort.py - PyCharm Community Edition 4.0.6
File Edit View Navigate Code Refactor Run Tools VCS Window Help
Project sorting (C:\Users\TARUNCH\PycharmProjects\sorting)
  mergesort.py
  heapsort.py
  mergesort.py
  quicksort.py
  quicksort2.py
  quicksortmed.py
  quicksortmedian.py
  sam.py
  External Libraries
mergesort.py x quicksortmedian.py x heapsort.py x quicksortmed.py x
a[k] = c[j]
j = j + 1
k = k + 1
while i < len(b):
    a[k] = b[i]
    i = i + 1
    k = k + 1
while j < len(c):
    a[k] = c[j]
    j = j + 1
    k = k + 1
st = time.time()
a = random.sample(range(1,1000),999)
a.sort()
msort(a)
print ("%s sec" , (time.time()-st)*1000)
Run: new new mergesort
C:\Python27\python.exe C:/Users/TARUNCH/PycharmProjects/sorting/mergesort.py
('%s sec', 10.999917984008789)
Process finished with exit code 0
22:32 05-06-2015
```

Merge sort execution: (Sorted case)



```
sorting - [C:\Users\TARUNCH\PycharmProjects\sorting] - ... \mergesort.py - PyCharm Community Edition 4.0.6
File Edit View Navigate Code Refactor Run Tools VCS Window Help
Project sorting (C:\Users\TARUNCH\PycharmProjects\sorting)
  mergesort.py
  heapsort.py
  mergesort.py
  quicksort.py
  quicksort2.py
  quicksortmed.py
  quicksortmedian.py
  sam.py
  External Libraries
mergesort.py x quicksortmedian.py x heapsort.py x quicksortmed.py x
a[k] = c[j]
j = j + 1
k = k + 1
while i < len(b):
    a[k] = b[i]
    i = i + 1
    k = k + 1
while j < len(c):
    a[k] = c[j]
    j = j + 1
    k = k + 1
st = time.time()
a = random.sample(range(1,1000),999)
a.sort()
msort(a)
print ("%s sec" , (time.time()-st)*1000)
Run: new new mergesort
C:\Python27\python.exe C:/Users/TARUNCH/PycharmProjects/sorting/mergesort.py
('%s sec', 7.999897003173828)
Process finished with exit code 0
22:32 05-06-2015
```


Merge sort: (Reversely sorted)



The screenshot shows the PyCharm IDE with a project named 'sorting'. The file explorer on the left lists several Python files: heapsort.py, mergesort.py, quicksort.py, quicksort2.py, quicksortmed.py, quicksortmedian.py, and sam.py. The 'mergesort.py' file is open in the editor. The code implements a merge sort algorithm. It starts by creating a random array 'a' of size 1000, sorted in reverse order using 'a.sort(reverse=True)'. The algorithm then recursively splits the array and merges it back in sorted order. The execution output at the bottom shows the program running successfully with an exit code of 0, and a timing message: ('%s sec', 9.00062942504883).

```
sorting - [C:\Users\TARUNCH\PycharmProjects\sorting] - ...mergesort.py - PyCharm Community Edition 4.0.6
File Edit View Navigate Code Refactor Run Tools VCS Window Help
Project sorting
mergesort.py
quicksortmedian.py
heapsort.py
quicksort2.py
quicksortmed.py
quicksortmedian.py
sam.py
External Libraries

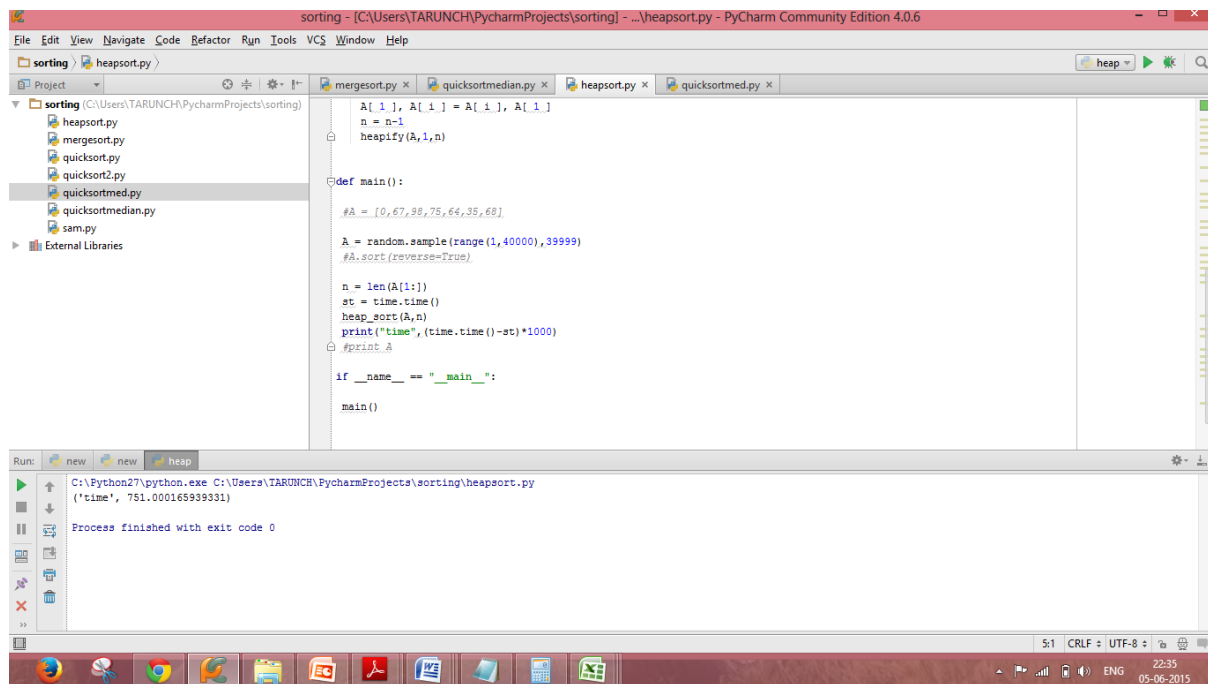
a[k] = c[j]
j = j + 1
k = k + 1

while i < len(b):
    a[k] = b[i]
    i = i + 1
    k = k + 1
while j < len(c):
    a[k] = c[j]
    j = j + 1
    k = k + 1

st = time.time()
a = random.sample(range(1,1000),999)
a.sort(reverse=True)
msort(a)
print ("%s sec", (time.time()-st)*1000)

Run: new new mergesort
C:\Python27\python.exe C:\Users\TARUNCH\PycharmProjects\sorting\mergesort.py
('%s sec', 9.00062942504883)
Process finished with exit code 0
5:1 CRLF UTF-8 22:33 05-06-2015
```

Heap sort: (Unsorted)



The screenshot shows the PyCharm IDE with the same 'sorting' project. The file explorer on the left is the same. The 'heapsort.py' file is open in the editor. The code implements a heap sort algorithm. It starts by creating a random array 'A' of size 40000. The algorithm then uses 'heapify' to convert the array into a max heap and repeatedly extracts the maximum element. The execution output at the bottom shows the program running successfully with an exit code of 0, and a timing message: ('time', 751.000165939331).

```
sorting - [C:\Users\TARUNCH\PycharmProjects\sorting] - ...heapsort.py - PyCharm Community Edition 4.0.6
File Edit View Navigate Code Refactor Run Tools VCS Window Help
Project sorting
heapsort.py
mergesort.py
quicksort.py
quicksort2.py
quicksortmed.py
quicksortmedian.py
sam.py
External Libraries

A[1], A[1] = A[1], A[1]
n = n-1
heapify(A,1,n)

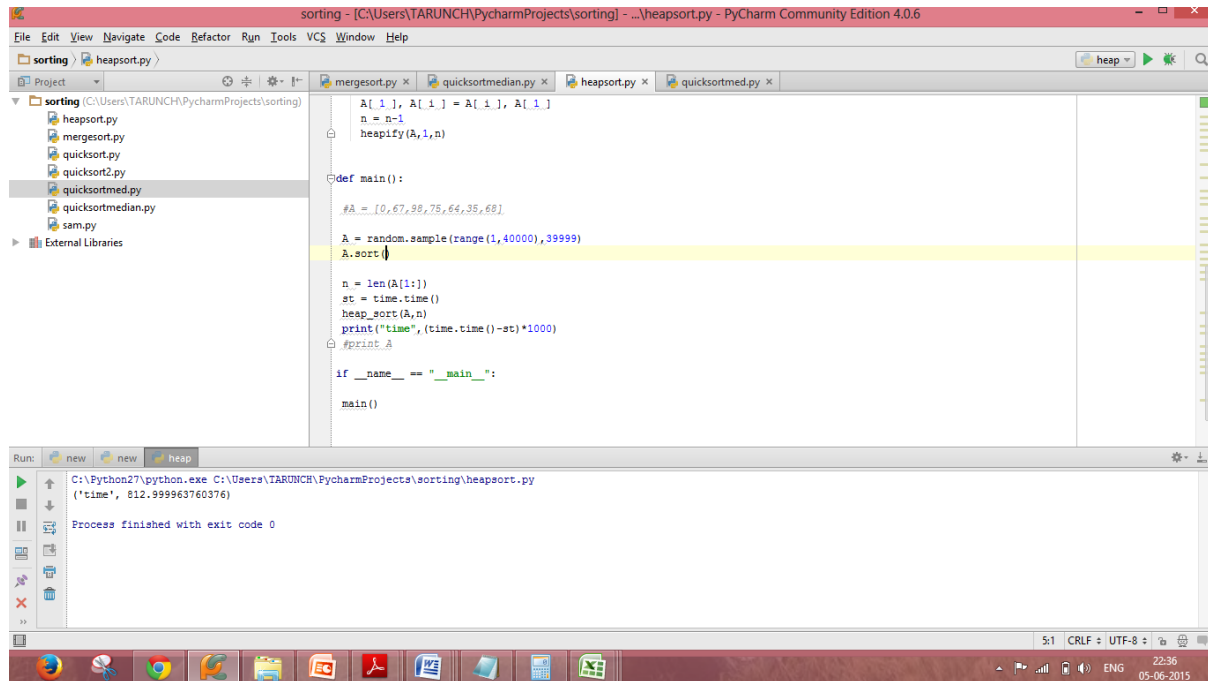
def main():
    #A = [0,67,38,75,64,35,68]
    A = random.sample(range(1,40000),39999)
    #A.sort(reverse=True)

    n = len(A[1:])
    st = time.time()
    heap_sort(A,n)
    print("time", (time.time()-st)*1000)
    #print A

if __name__ == "__main__":
    main()

Run: new new heap
C:\Python27\python.exe C:\Users\TARUNCH\PycharmProjects\sorting\heapsort.py
('time', 751.000165939331)
Process finished with exit code 0
5:1 CRLF UTF-8 22:35 05-06-2015
```

Heap sort(sorted):



The screenshot shows the PyCharm IDE with a project named 'sorting'. The file explorer on the left lists several Python files: heap_sort.py, mergesort.py, quicksort.py, quicksort2.py, quicksortmed.py, quicksortmedian.py, and sam.py. The 'heap_sort.py' file is open in the editor, showing the following code:

```
A[1:], A[1:] = A[1:], A[1:]
n = n-1
heapify(A,1,n)

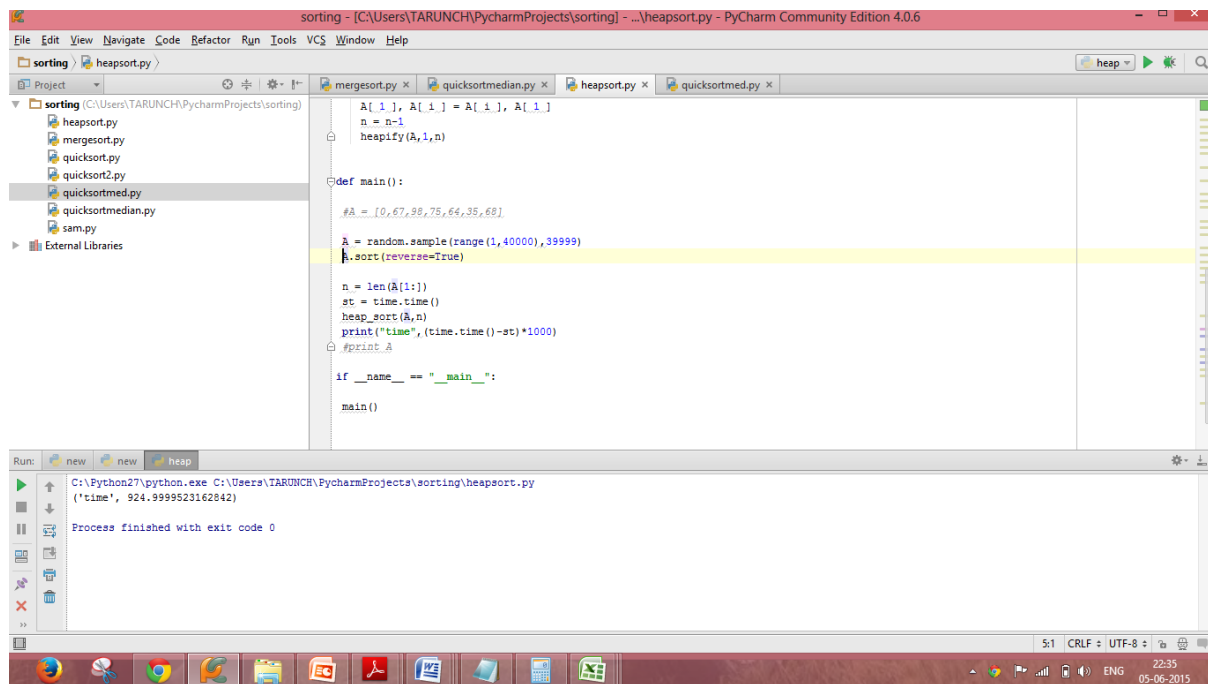
def main():
    #A = [0,67,38,75,64,35,68]
    A = random.sample(range(1,40000),39999)
    A.sort()

    n = len(A[1:])
    st = time.time()
    heap_sort(A,n)
    print("time", (time.time()-st)*1000)
    #print A

if __name__ == "__main__":
    main()
```

The Run window at the bottom shows the execution command: `C:\Python27\python.exe C:\Users\TARUNCH\PycharmProjects\sorting\heap_sort.py`. The output is: `(*time', 812.999963760376)`. The process finished with exit code 0.

Heap sort(Reversely sorted):



The screenshot shows the PyCharm IDE with the same project 'sorting'. The 'heap_sort.py' file is open, but the code is modified to sort in reverse order:

```
A[1:], A[1:] = A[1:], A[1:]
n = n-1
heapify(A,1,n)

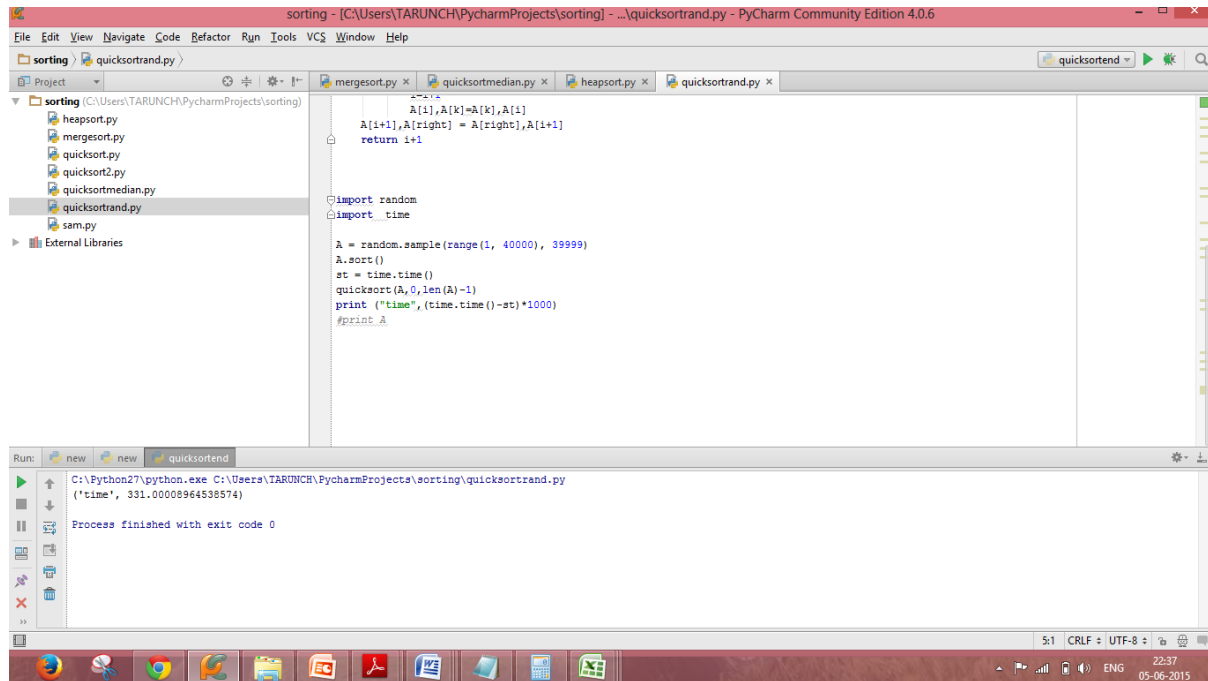
def main():
    #A = [0,67,38,75,64,35,68]
    A = random.sample(range(1,40000),39999)
    A.sort(reverse=True)

    n = len(A[1:])
    st = time.time()
    heap_sort(A,n)
    print("time", (time.time()-st)*1000)
    #print A

if __name__ == "__main__":
    main()
```

The Run window shows the execution command: `C:\Python27\python.exe C:\Users\TARUNCH\PycharmProjects\sorting\heap_sort.py`. The output is: `(*time', 924.9999523162842)`. The process finished with exit code 0.

Quick sort(sorted):



The screenshot shows the PyCharm IDE with the file `quicksortrand.py` open. The code defines a `quicksort` function and a `main` function. The `main` function generates a random array `A` of 40,000 elements and sorts it using `quicksort`. The execution results at the bottom show the output: `('time', 331.00008964538574)` and `Process finished with exit code 0`.

```
def quicksort(A, left, right):
    if left < right:
        p = partition(A, left, right)
        quicksort(A, left, p-1)
        quicksort(A, p+1, right)
    return A

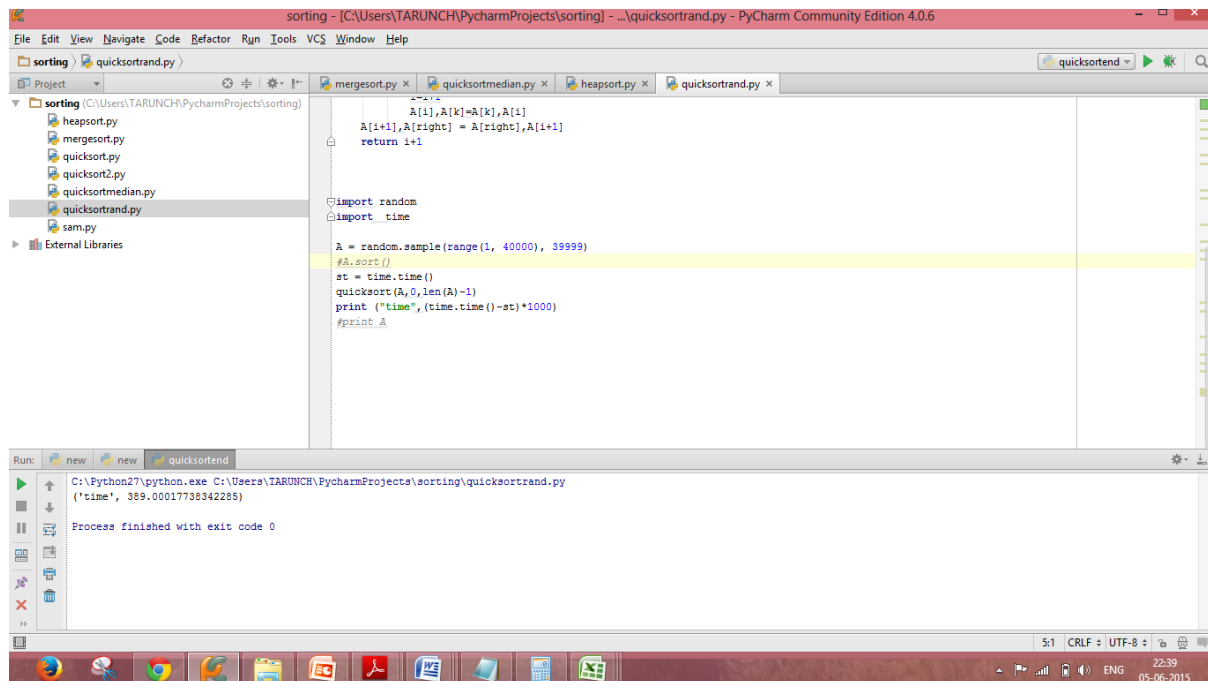
def partition(A, left, right):
    pivot = A[right]
    i = left
    for j in range(left, right):
        if A[j] < pivot:
            A[i], A[j] = A[j], A[i]
            i += 1
    A[i], A[right] = A[right], A[i]
    return i

import random
import time

A = random.sample(range(1, 40000), 39999)
A.sort()
st = time.time()
quicksort(A, 0, len(A)-1)
print("time", (time.time()-st)*1000)
#print A
```

Run: C:\Python27\python.exe C:\Users\TARUNCH\PycharmProjects\sorting\quicksortrand.py ('time', 331.00008964538574) Process finished with exit code 0

Quick sort (Unsorted):



The screenshot shows the PyCharm IDE with the file `quicksortrand.py` open. The code is identical to the previous screenshot, but the `A.sort()` line is commented out. The execution results at the bottom show the output: `('time', 389.00017738342285)` and `Process finished with exit code 0`.

```
def quicksort(A, left, right):
    if left < right:
        p = partition(A, left, right)
        quicksort(A, left, p-1)
        quicksort(A, p+1, right)
    return A

def partition(A, left, right):
    pivot = A[right]
    i = left
    for j in range(left, right):
        if A[j] < pivot:
            A[i], A[j] = A[j], A[i]
            i += 1
    A[i], A[right] = A[right], A[i]
    return i

import random
import time

A = random.sample(range(1, 40000), 39999)
#A.sort()
st = time.time()
quicksort(A, 0, len(A)-1)
print("time", (time.time()-st)*1000)
#print A
```

Run: C:\Python27\python.exe C:\Users\TARUNCH\PycharmProjects\sorting\quicksortrand.py ('time', 389.00017738342285) Process finished with exit code 0

Quick sort(Unsorted) median as pivot:

The screenshot shows the PyCharm IDE with a project named 'sorting'. The file explorer on the left lists several Python files: heapsort.py, mergesort.py, quicksort.py, quicksort2.py, quicksortmedian.py, quicksortrand.py, and sam.py. The 'quicksortmedian.py' file is open in the editor. The code defines a 'split' function that uses the median of the current subarray as a pivot. The 'quicksort' function is recursive, calling 'split' to partition the array. The main part of the code generates a random array 'A' of size 10,000 and calls 'quicksort(A, 0, len(A) - 1)'. The Run window at the bottom shows the execution output: ('time', 56.999921798706055) and 'Process finished with exit code 0'.

```
def split(A, left, right):
    pivot = A[median(A, left, right)]
    i = left - 1
    for k in range(left, right - 1, 1):
        if A[k] <= pivot:
            i = i + 1
            A[i], A[k] = A[k], A[i]
    A[i + 1], A[right - 1] = A[right - 1], A[i + 1]
    return i + 1

A = random.sample(range(1, 10000), 9999)
#A.sort()
st = time.time()
quicksort(A, 0, len(A) - 1)
#print A
```

Run: C:\Python27\python.exe C:/Users/TARUNCH/PycharmProjects/sorting/quicksortmedian.py ('time', 56.999921798706055)
Process finished with exit code 0

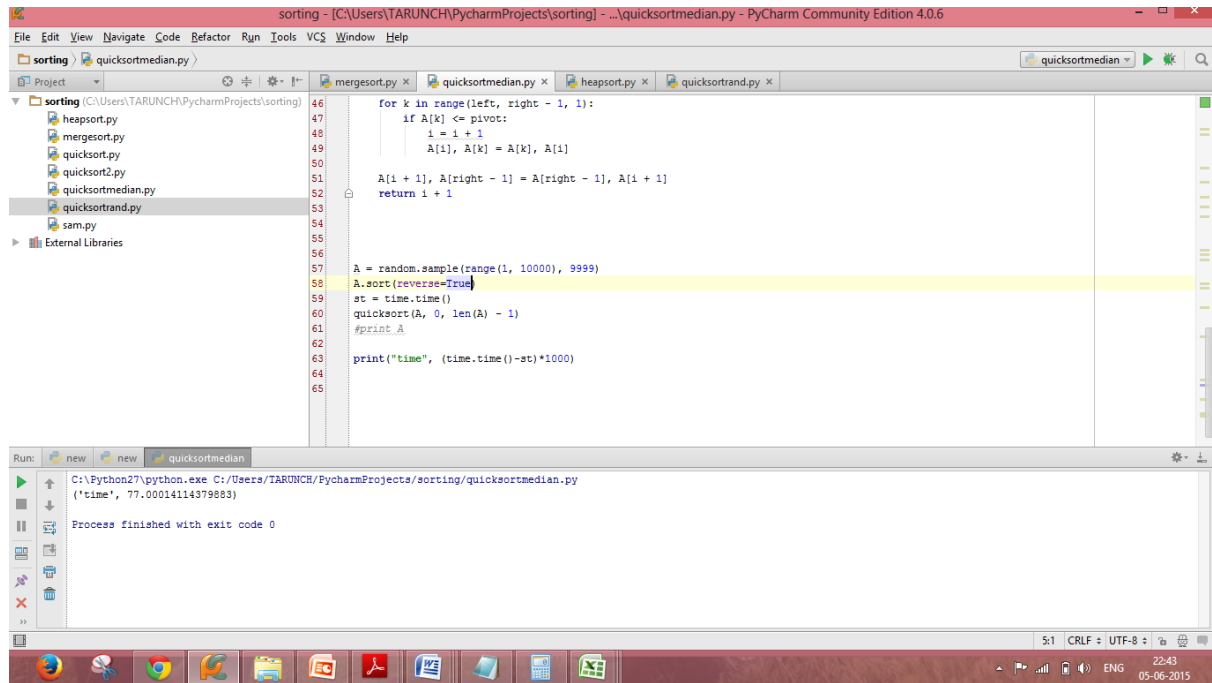
Quick sort(Sorted) median as pivot:

The screenshot shows the PyCharm IDE with the same project 'sorting'. The 'quicksortmedian.py' file is open. The code is similar to the previous one, but the 'split' function is not shown in the visible code block. The main part of the code generates a random array 'A' of size 10,000, sorts it, and then calls 'quicksort(A, 0, len(A) - 1)'. The Run window at the bottom shows the execution output: ('time', 40.9998893737793) and 'Process finished with exit code 0'.

```
A = random.sample(range(1, 10000), 9999)
A.sort()
st = time.time()
quicksort(A, 0, len(A) - 1)
#print A
print("time", (time.time()-st)*1000)
```

Run: C:\Python27\python.exe C:/Users/TARUNCH/PycharmProjects/sorting/quicksortmedian.py ('time', 40.9998893737793)
Process finished with exit code 0

Quick sort(Reversely sorted) median as pivot:



```
46     for k in range(left, right - 1, 1):
47         if A[k] <= pivot:
48             i = i + 1
49             A[i], A[k] = A[k], A[i]
50
51     A[i + 1], A[right - 1] = A[right - 1], A[i + 1]
52     return i + 1
53
54
55
56
57 A = random.sample(range(1, 10000), 9999)
58 A.sort(reverse=True)
59 st = time.time()
60 quicksort(A, 0, len(A) - 1)
61 #print A
62
63 print("time", (time.time() - st) * 1000)
64
65
```

Run: C:\Python27\python.exe C:/Users/TARUNCH/PycharmProjects/sorting/quicksortmedian.py ('time', 77.00014114379883)

Process finished with exit code 0