# sorting

November 20, 2021

## 1 Sorting Comparison

[ ]:
```
from comparison import *
from sorting import *
import numpy as np
import matplotlib.pyplot as plt

np.random.seed(42)
```

### 1.1 Code / Unit Test

There are 3 types of test:

1) Sorted array
2) Reverse sorted array
3) Random input

**NOTE** that assertions are hidden in Comparison

[3]:
```
comparisons = Comparison()
```

[3]:
```
sorted_case = [i for i in range(1000)]
reverse_sorted_case = [i for i in reversed(range(1000))]
```

---

#### 1.1.1 Insertion Sort

[4]:
```
_ = comparisons.insertionSort(numbers = sorted_case)
```

[7]:
```
_ = comparisons.insertionSort(numbers = reverse_sorted_case)
```

[6]:
```
for i in range(1000):
    random = list(np.random.randint(-100000, 100000, i))
    _ = comparisons.insertionSort(numbers = random)
```

---

### 1.1.2 Merge Sort

[9]:
```
_ = comparisons.mergeSort(numbers = sorted_case)
```

[10]:
```
_ = comparisons.mergeSort(numbers = reverse_sorted_case)
```

[11]:
```
for i in range(1000):
    random = list(np.random.randint(-100000, 100000, i))
    _ = comparisons.mergeSort(numbers = random)
```

---

### 1.1.3 Quick Sort

**Vanilla**

[4]:
```
_ = comparisons.quickSortVanilla(numbers = sorted_case)
```

[5]:
```
_ = comparisons.quickSortVanilla(numbers = reverse_sorted_case)
```

[6]:
```
for i in range(1000):
    random = list(np.random.randint(-100000, 100000, i))
    _ = comparisons.quickSortVanilla(numbers = random)
```

**Randomized**

[4]:
```
_ = comparisons.quickSortRandomized(numbers = sorted_case)
```

[5]:
```
_ = comparisons.quickSortRandomized(numbers = reverse_sorted_case)
```

[6]:
```
for i in range(1000):
    random = list(np.random.randint(-100000, 100000, i))
    _ = comparisons.quickSortRandomized(numbers = random)
```

---

### 1.1.4 Heap

**Build Heap**

[18]:
```
for i in range(0, 16, 5):
    random = list(np.random.randint(-500, 500, i))
    heap = MinHeap(random)
    print(f"Before: {random} After: {heap}")
```

```
Before: [] After: [None]
Before: [-263, 43, 67, -322, -432] After: [None, -432, -322, 67, -263, 43]
Before: [222, 397, 62, 367, 321, 347, 481, -11, -6, 425] After: [None, -11, -6,
62, 222, 321, 347, 481, 367, 397, 425]
Before: [402, -214, -315, 117, 210, -289, -170, 474, 58, -159, 135, -155, -386,
```

```
110, -343] After: [None, -386, -214, -343, 58, -159, -315, -170, 474, 117, 210,
135, -155, -289, 110, 402]
```

**Heap Sort**

[19]:
```
_ = comparisons.heapSort(numbers = sorted_case)
```

[20]:
```
_ = comparisons.heapSort(numbers = reverse_sorted_case)
```

[21]:
```
for i in range(1000):
    random = list(np.random.randint(-100000, 100000, i))
    _ = comparisons.heapSort(numbers = random)
```

---

## 1.2 Analysis

Generate a set of inputs. When Comparison is called with the input set, each sorting algorithm is ran and the runtime is recorded. On exit, a graph of the runtime is plotted.

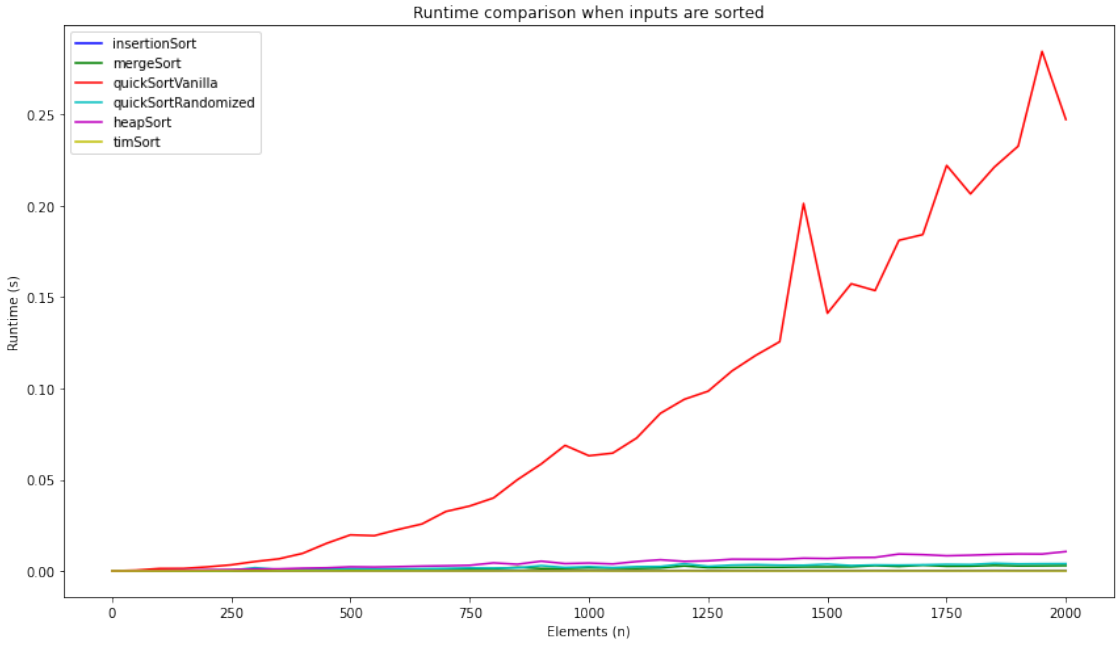
[7]:
```
comparisons = Comparison("graphs")
```

### 1.2.1 Sorted test

[8]:
```
sorted_input = []

for element in range(0, 2001, 50):
    random = list(range(element))
    sorted_input.append(random)

runtime = comparisons(sorted_input, "Runtime comparison when inputs are␣
 ↪sorted", "sorted_input")
```




```
Saving output to graphs/sorted_input
```

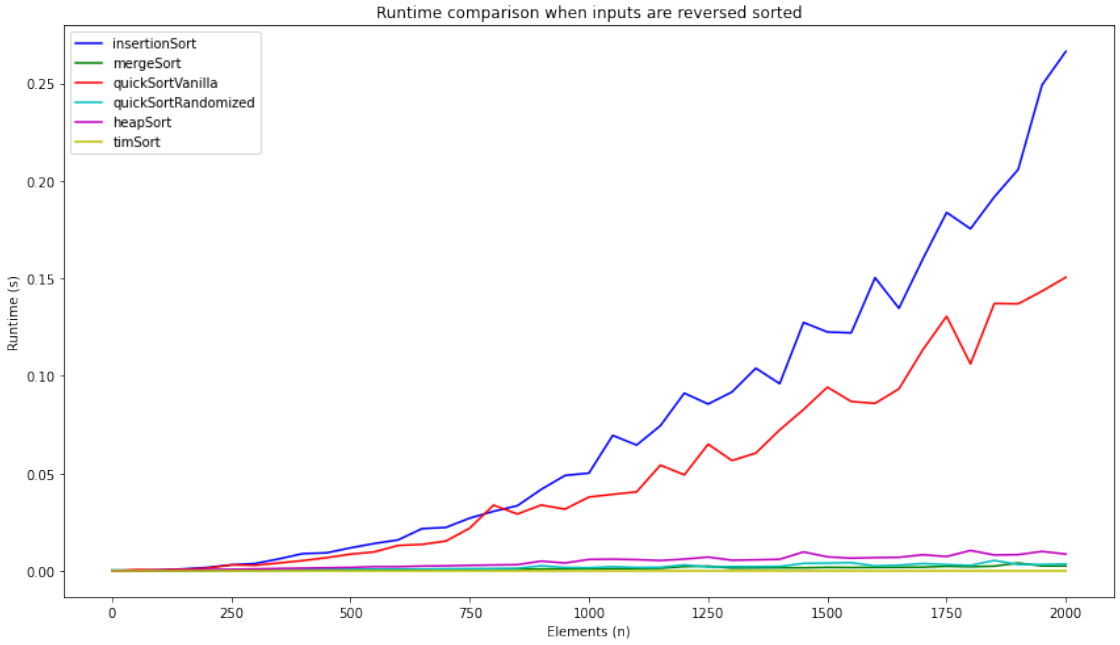
---

### 1.2.2 Reversed sorted test

[4]:
```
reversed_input = []

for element in range(0, 2001, 50):
    random = list(reversed(range(element)))
    reversed_input.append(random)

runtime = comparisons(reversed_input, "Runtime comparison when inputs are␣
 ↪reversed sorted", "reversed_sorted_input")
```




```
Expected to save graph but provided output directory of None
```

---

### 1.2.3 Heap sort analysis

[10]:
```
sorted_heap = []
for i in range(0, 10001, 50):
    comparisons.heapSort(numbers = list(range(i)))
    sorted_heap.append(comparisons.runtime["heapSort"][-1])

comparisons.runtime = {}
```

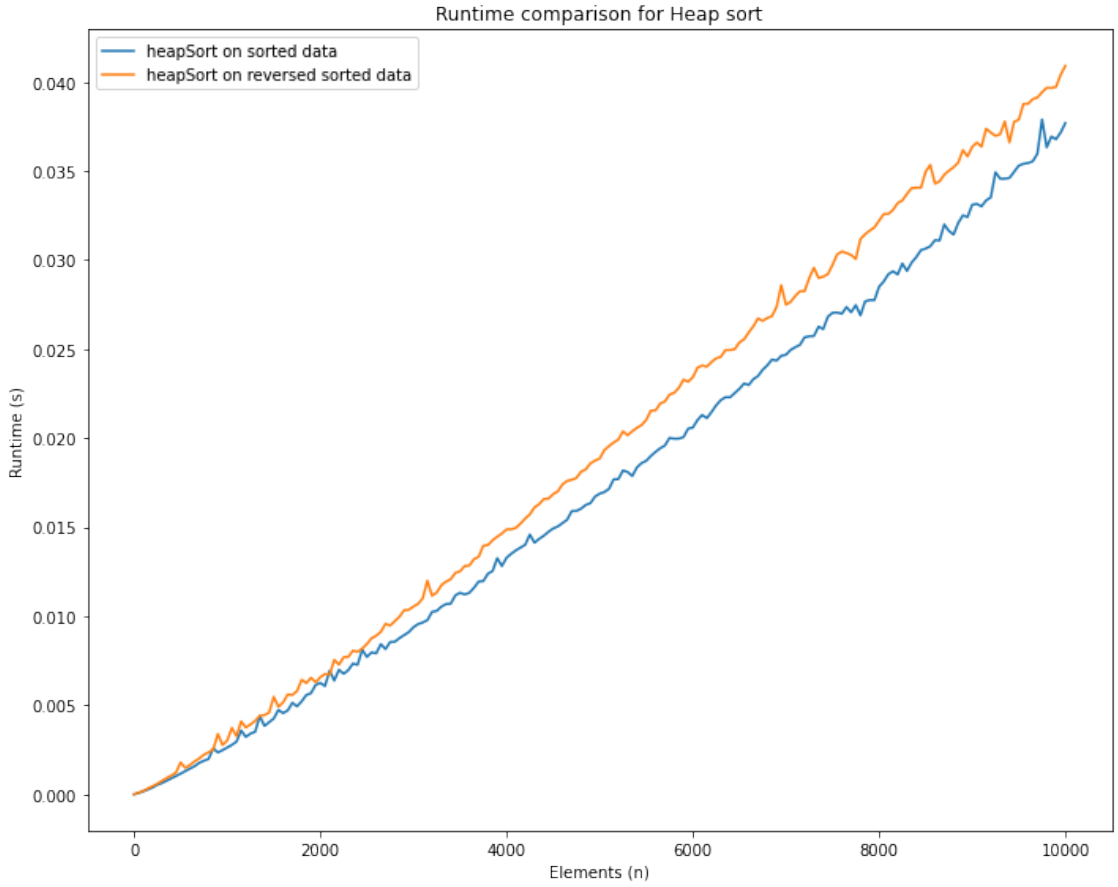
[31]:
```
reversed_heap = []
for i in range(0, 10001, 50):
    comparisons.heapSort(numbers = list(reversed(range(i))))
    reversed_heap.append(comparisons.runtime["heapSort"][-1])

comparisons.runtime = {}
```

[32]:
```
fig, ax = plt.subplots(figsize = (10, 8))

for label, data in zip(["heapSort on sorted data", "heapSort on reversed sorted␣
 ↪data"], [sorted_heap, reversed_heap]):
    ax.plot(list(range(0, 10001, 50)), data, label = label)
```

```
ax.set(title = "Runtime comparison for Heap sort", xlabel = "Elements (n)",␣
 ↪ylabel = "Runtime (s)")
ax.legend(loc = "best")
plt.tight_layout()
plt.show()
fig.savefig("graphs/heap_comparison")
```



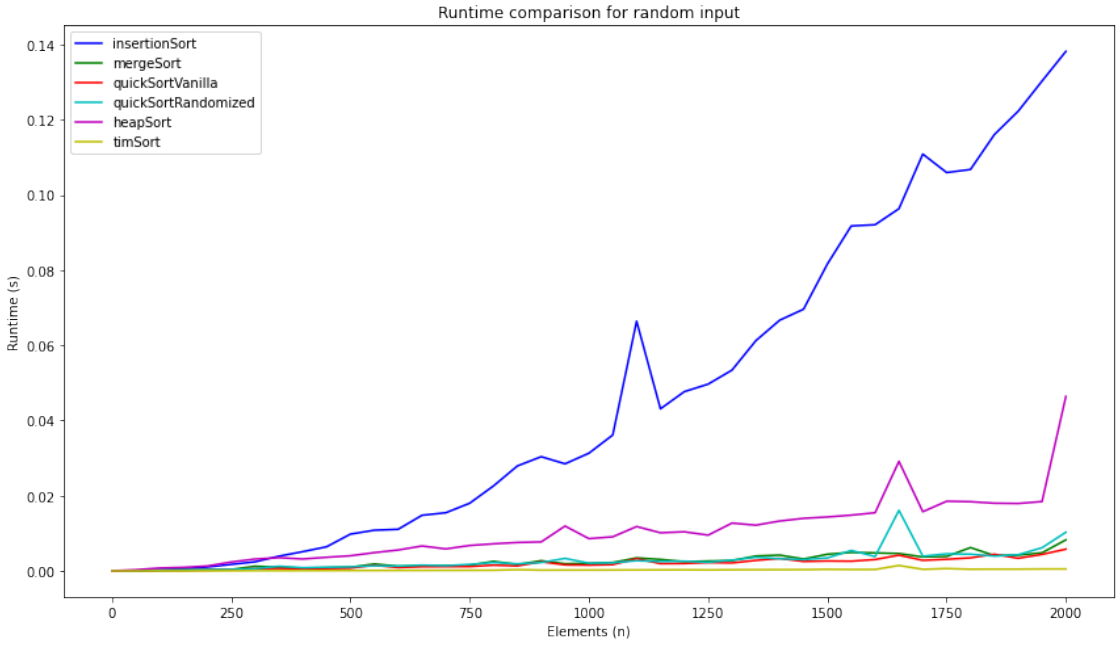

---

### 1.2.4 Random input

[11]:
```
random_input = []

for element in range(0, 2001, 50):
    random = list(np.random.randint(-100000, 100000, element))
    random_input.append(random)

runtime = comparisons(random_input, "Runtime comparison for random input",␣
 ↪"random_input")
```




```
Saving output to graphs/random_input
```

---

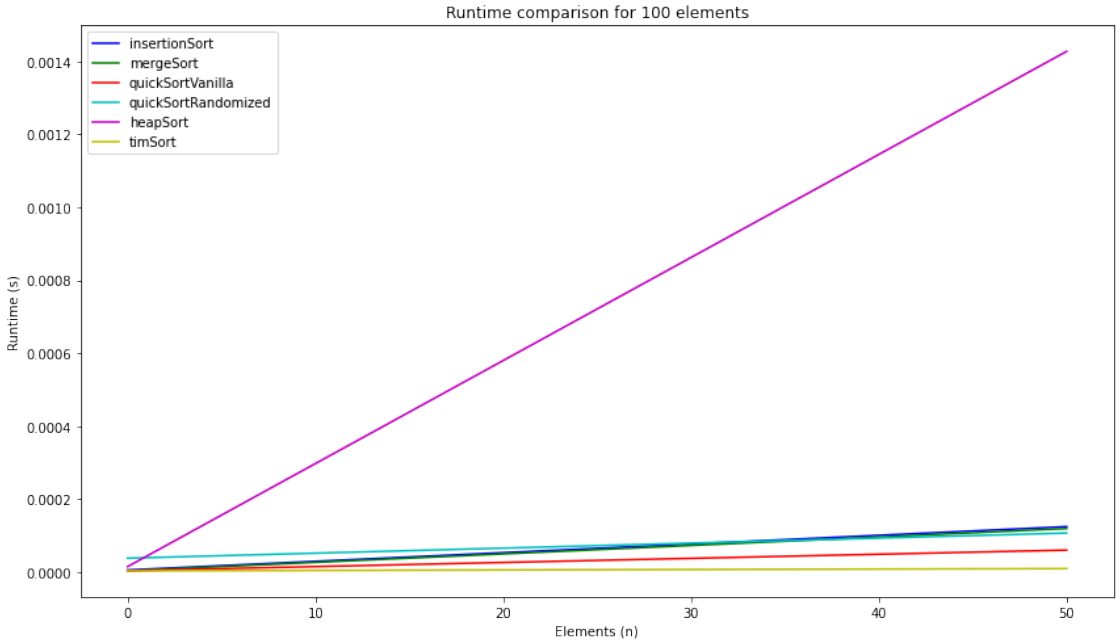
### 1.2.5 Small Input

### 1.2.6 100 elements

[12]:
```
small_100 = []

for element in range(0, 51, 50):
    random = list(np.random.randint(-100000, 100000, element))
    small_100.append(random)

runtime = comparisons(small_100, "Runtime comparison for 100 elements",␣
↪"100_element")
```



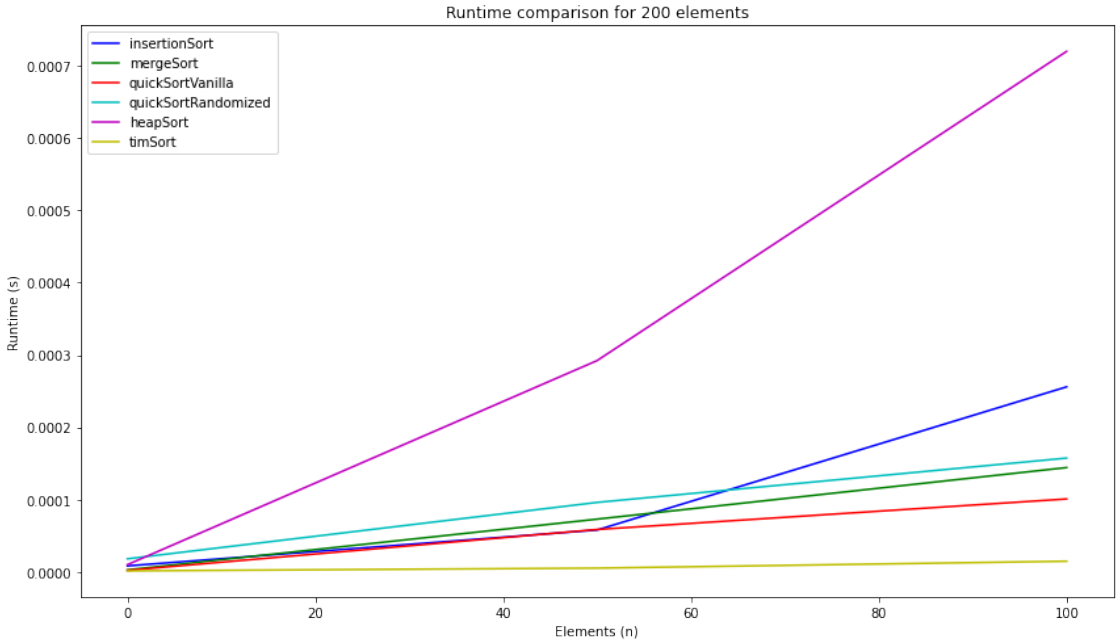

```
Saving output to graphs/100_element
```

### 200 elements

[13]:
```
small_200 = []

for element in range(0, 101, 50):
    random = list(np.random.randint(-100000, 100000, element))
    small_200.append(random)

runtime = comparisons(small_200, "Runtime comparison for 200 elements",␣
 ↪"200_element")
```



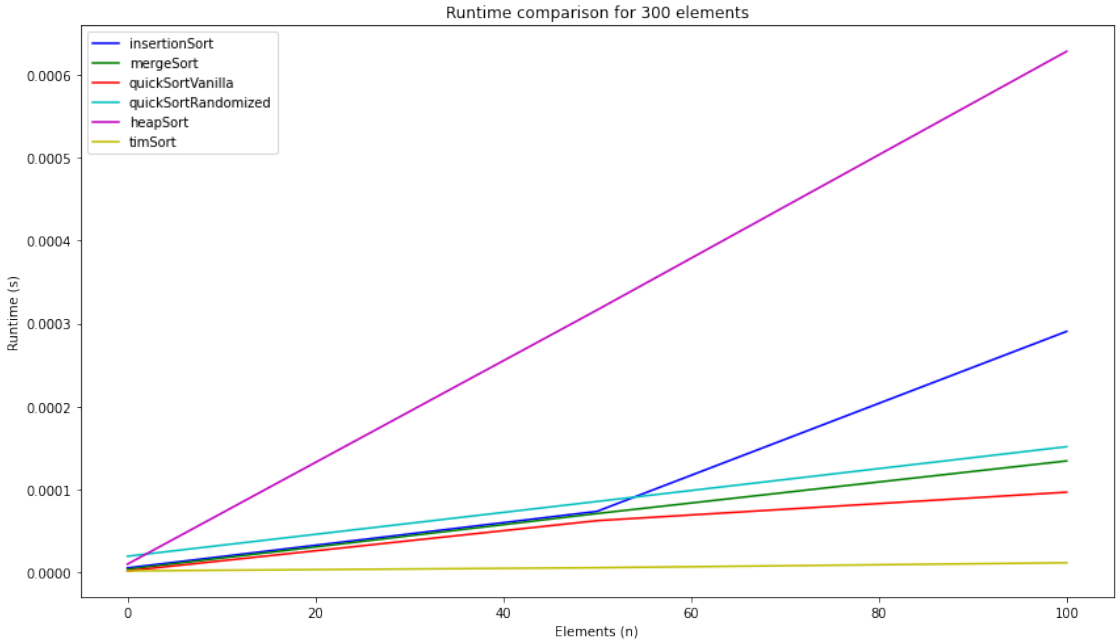

```
Saving output to graphs/200_element
```

### 300 elements

[14]:
```
small_300 = []

for element in range(0, 101, 50):
    random = list(np.random.randint(-100000, 100000, element))
    small_300.append(random)

runtime = comparisons(small_300, "Runtime comparison for 300 elements",␣
 ↪"300_element")
```



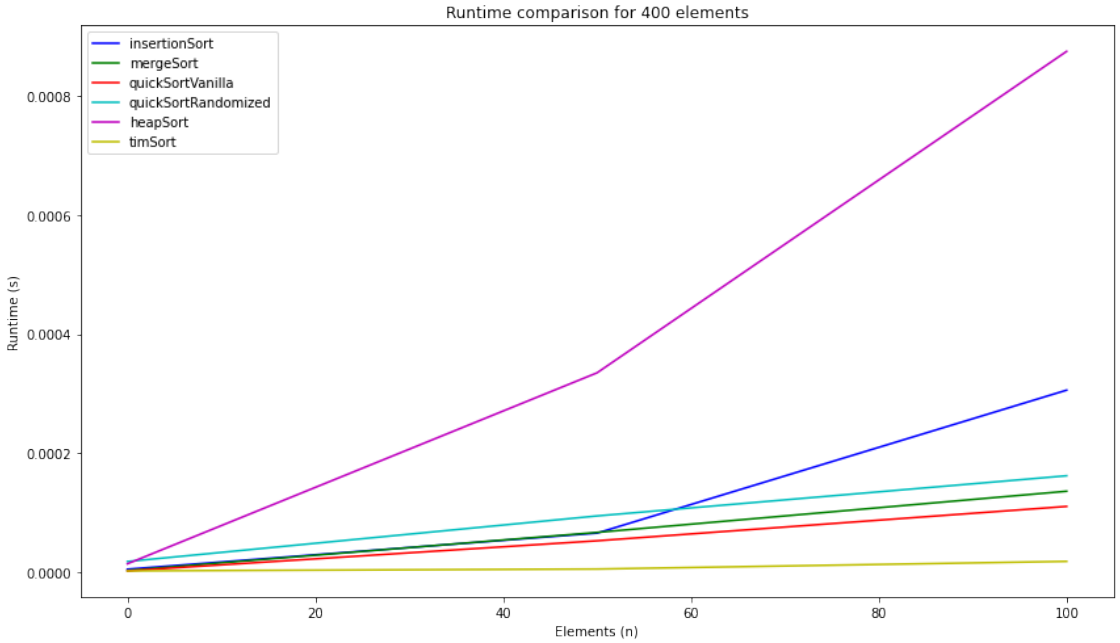

```
Saving output to graphs/300_element
```

### Small 400

[15]:
```
small_400 = []

for element in range(0, 101, 50):
    random = list(np.random.randint(-100000, 100000, element))
    small_400.append(random)

runtime = comparisons(small_400, "Runtime comparison for 400 elements",␣
 ↪"400_element")
```



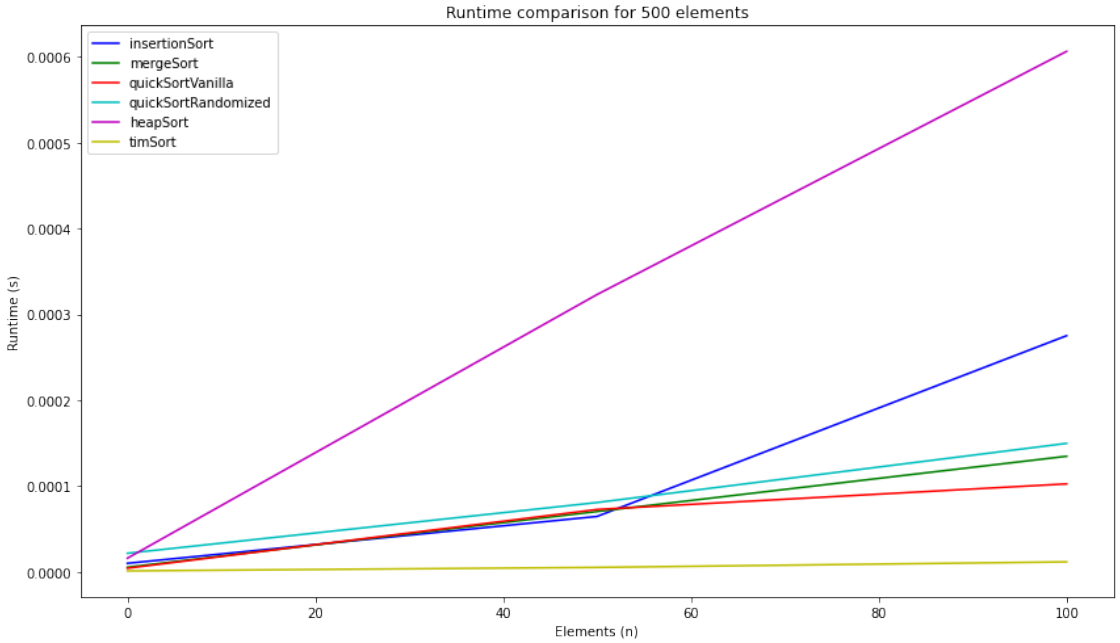

```
Saving output to graphs/400_element
```

### Small 500

[17]:
```
small_500 = []

for element in range(0, 101, 50):
    random = list(np.random.randint(-100000, 100000, element))
    small_500.append(random)

runtime = comparisons(small_500, "Runtime comparison for 500 elements",␣
 ↪"500_element")
```




```
Saving output to graphs/500_element
```