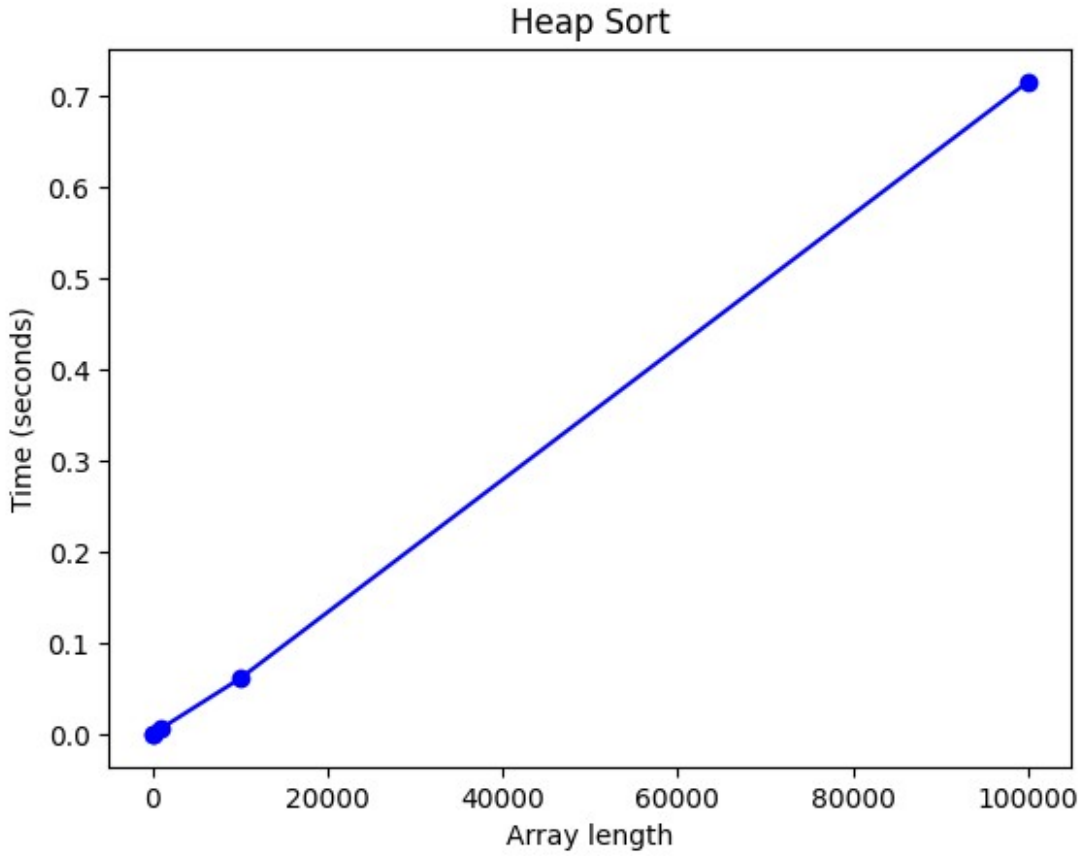
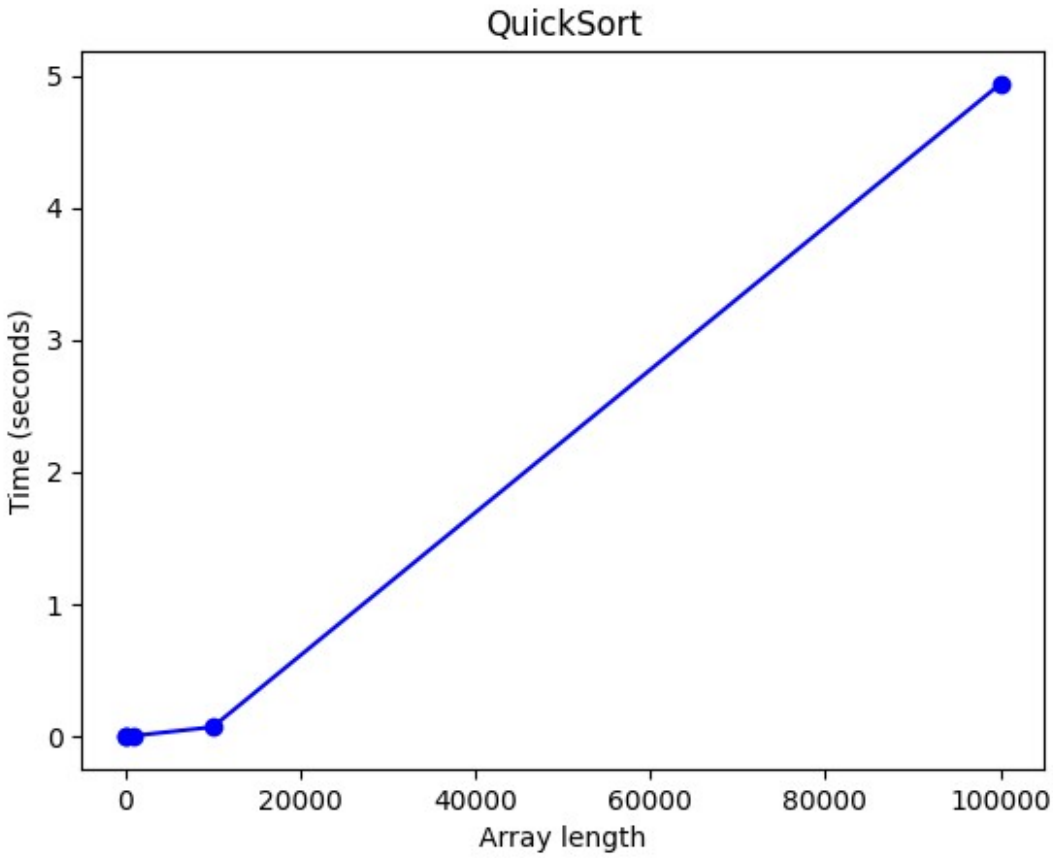
```python
from matplotlib import pyplot as plt
import random
import time
import sys

def heap_sort(arr):
    n = len(arr)

    # Build a max heap
    for i in range(n//2 - 1, -1, -1):
        heapify(arr, n, i)

    # Extract elements from the heap one by one
    for i in range(n-1, 0, -1):
        # Swap the root element (largest) with the last element
        arr[i], arr[0] = arr[0], arr[i]

        # Max heapify the reduced heap
        heapify(arr, i, 0)

def heapify(arr, n, i):
    # Initialize the largest as root
    largest = i
    left = 2*i + 1
    right = 2*i + 2

    # Check if left child of root exists and is greater than root
    if left < n and arr[left] > arr[largest]:
        largest = left

    # Check if right child of root exists and is greater than root
    if right < n and arr[right] > arr[largest]:
        largest = right

    # Change root, if needed
    if largest != i:
        arr[i], arr[largest] = arr[largest], arr[i]
        # Heapify the root
        heapify(arr, n, largest)


arr = []
times1 = []
cases = []
j = 1
for k in range(5):
    j *= 10
    for i in range(j):
        arr.append(random.randint(0, 100))
```

```
    start = time.time()
    heap_sort(arr)
    end = time.time()
    times1.append(end-start)
    cases.append(j)
# print("Sorted array is:", arr)

plt.plot(cases, times1, 'bo-')
plt.title('Heap Sort')
plt.xlabel('Array length')
plt.ylabel('Time (seconds)')
plt.show()
```




```
def quicksort(arr):
    if len(arr) <= 1:
        return arr
    else:
        pivot = arr[0]
        less = [x for x in arr[1:] if x <= pivot]
        greater = [x for x in arr[1:] if x > pivot]
        return quicksort(less) + [pivot] + quicksort(greater)

arr = []
times2 = []
```

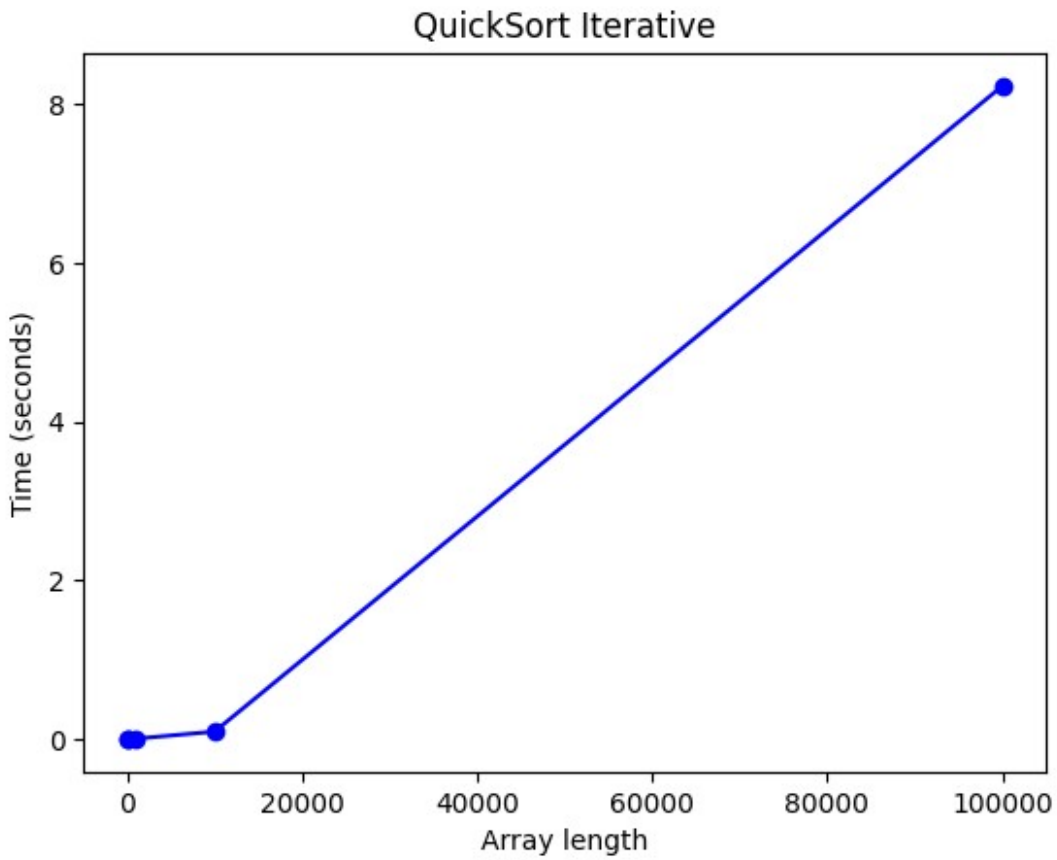
```
cases = []
j = 1
for k in range(5):
    j *= 10
    for i in range(j):
        arr.append(random.randint(0, 100))
    start = time.time()
    quicksort(arr)
    end = time.time()
    times2.append(end-start)
    cases.append(j)

plt.plot(cases, times2,'bo-')
plt.title('QuickSort')
plt.xlabel('Array length')
plt.ylabel('Time (seconds)')
plt.show()
```




```
# Python program for implementation of Quicksort

# This function is same in both iterative and recursive
def partition(arr,l,h):
    i = ( l - 1 )
    x = arr[h]
```

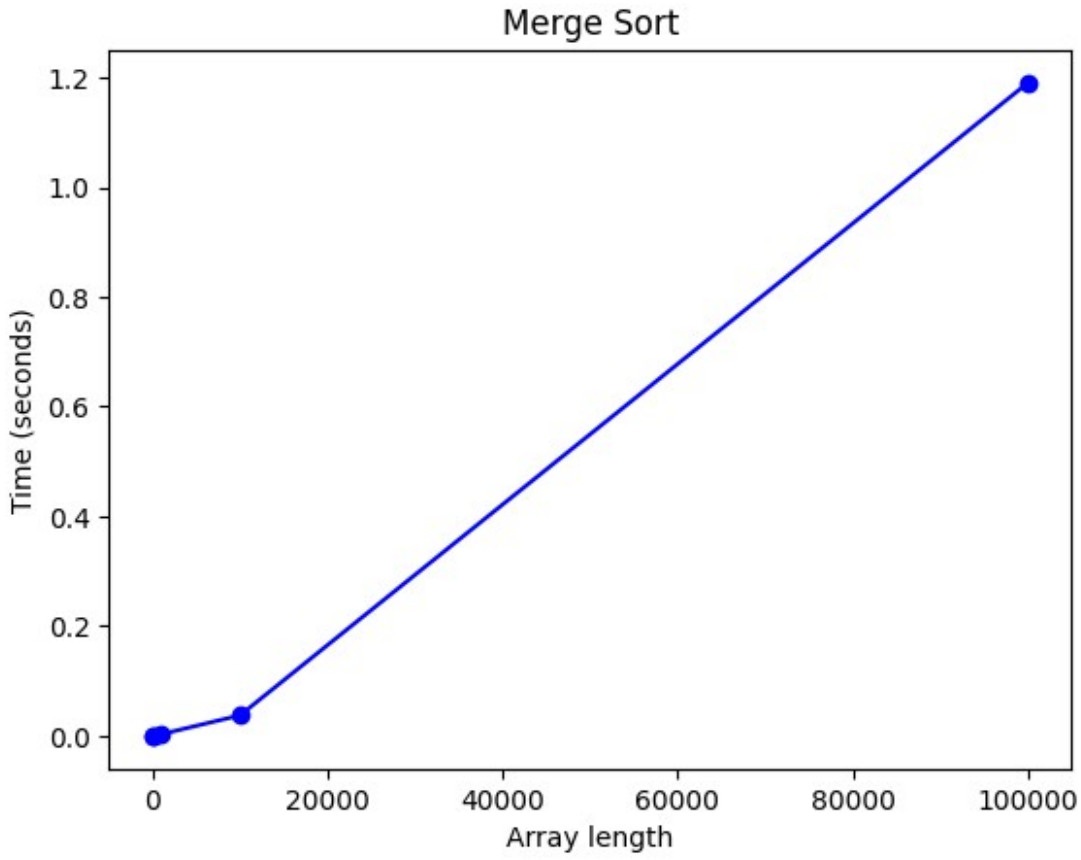
```
    for j in range(l , h):
        if   arr[j] <= x:

            # increment index of smaller element
            i = i+1
            arr[i],arr[j] = arr[j],arr[i]

    arr[i+1],arr[h] = arr[h],arr[i+1]
    return (i+1)

# Function to do Quick sort
# arr[] --> Array to be sorted,
# l  --> Starting index,
# h  --> Ending index
def quickSortIterative(arr,l,h):

    # Create an auxiliary stack
    size = h - l + 1
    stack = [0] * (size)

    # initialize top of stack
    top = -1

    # push initial values of l and h to stack
    top = top + 1
    stack[top] = l
    top = top + 1
    stack[top] = h

    # Keep popping from stack while is not empty
    while top >= 0:

        # Pop h and l
        h = stack[top]
        top = top - 1
        l = stack[top]
        top = top - 1

        # Set pivot element at its correct position in
        # sorted array
        p = partition( arr, l, h )

        # If there are elements on left side of pivot,
        # then push left side to stack
        if p-1 > l:
            top = top + 1
            stack[top] = l
            top = top + 1
            stack[top] = p - 1
```

```
        # If there are elements on right side of pivot,
        # then push right side to stack
        if p+1 < h:
            top = top + 1
            stack[top] = p + 1
            top = top + 1
            stack[top] = h


arr = []
times5 = []
cases = []
j = 1
for k in range(5):
    j *= 10
    for i in range(j):
        arr.append(random.randint(0, 100))
    start = time.time()
    quickSortIterative(arr, 0, j-1)
    end = time.time()
    times5.append(end-start)
    cases.append(j)

plt.plot(cases, times5,'bo-')
plt.title('QuickSort Iterative')
plt.xlabel('Array length')
plt.ylabel('Time (seconds)')
plt.show()
```

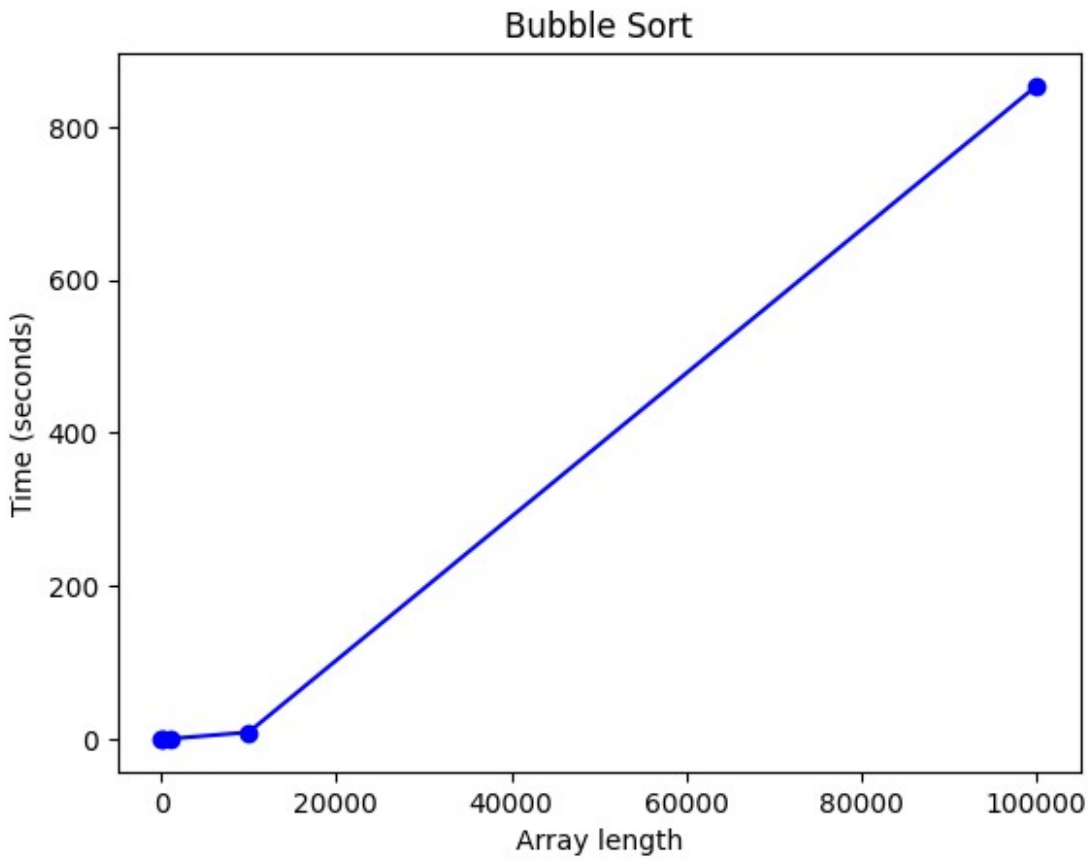



```python
def merge_sort(arr):
    if len(arr) <= 1:
        return arr

    mid = len(arr) // 2
    left = merge_sort(arr[:mid])
    right = merge_sort(arr[mid:])

    return merge(left, right)

def merge(left, right):
    result = []

    while left and right:
        if left[0] <= right[0]:
            result.append(left.pop(0))
        else:
            result.append(right.pop(0))

    if left:
        result.extend(left)
    if right:
        result.extend(right)
```

```
    return result

arr = []
times3 = [] #change
cases = []
j = 1
for k in range(5):
    j *= 10
    for i in range(j):
        arr.append(random.randint(0, 100))
    start = time.time()
    merge_sort(arr)    #change
    end = time.time()
    times3.append(end-start) #change
    cases.append(j)

plt.plot(cases, times3,'bo-') #change
plt.title('Merge Sort') #change
plt.xlabel('Array length')
plt.ylabel('Time (seconds)')
plt.show()
```

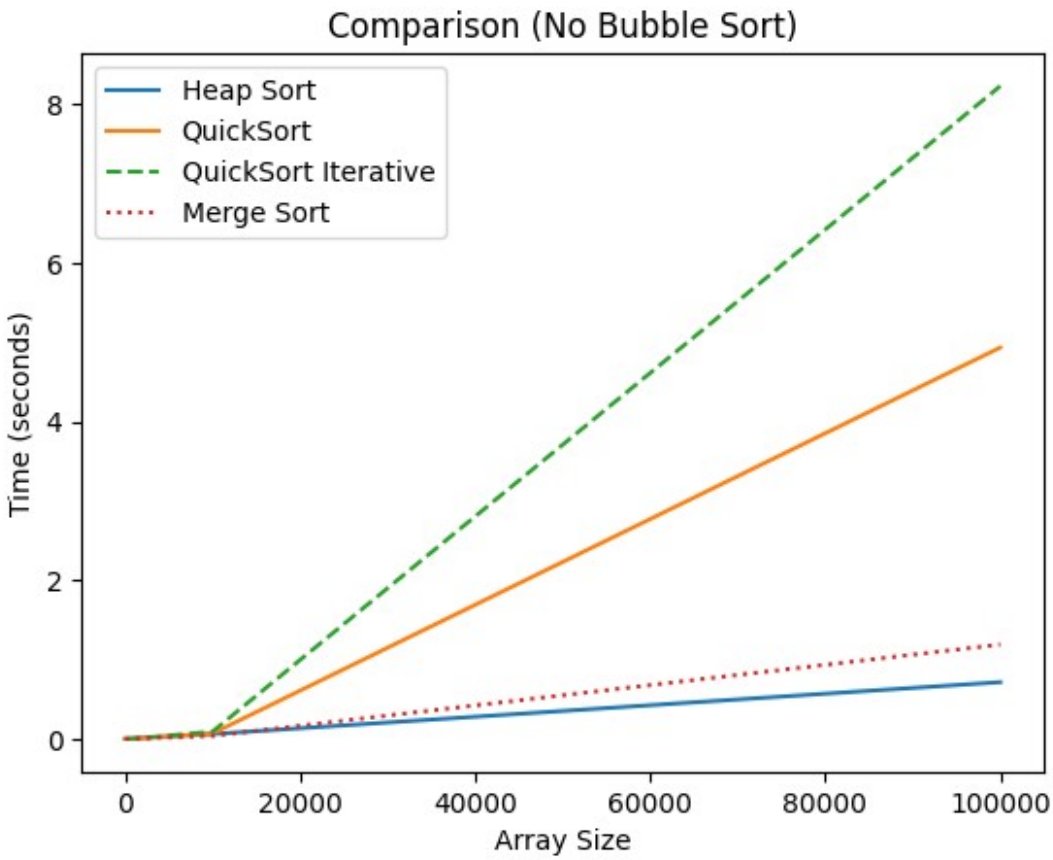


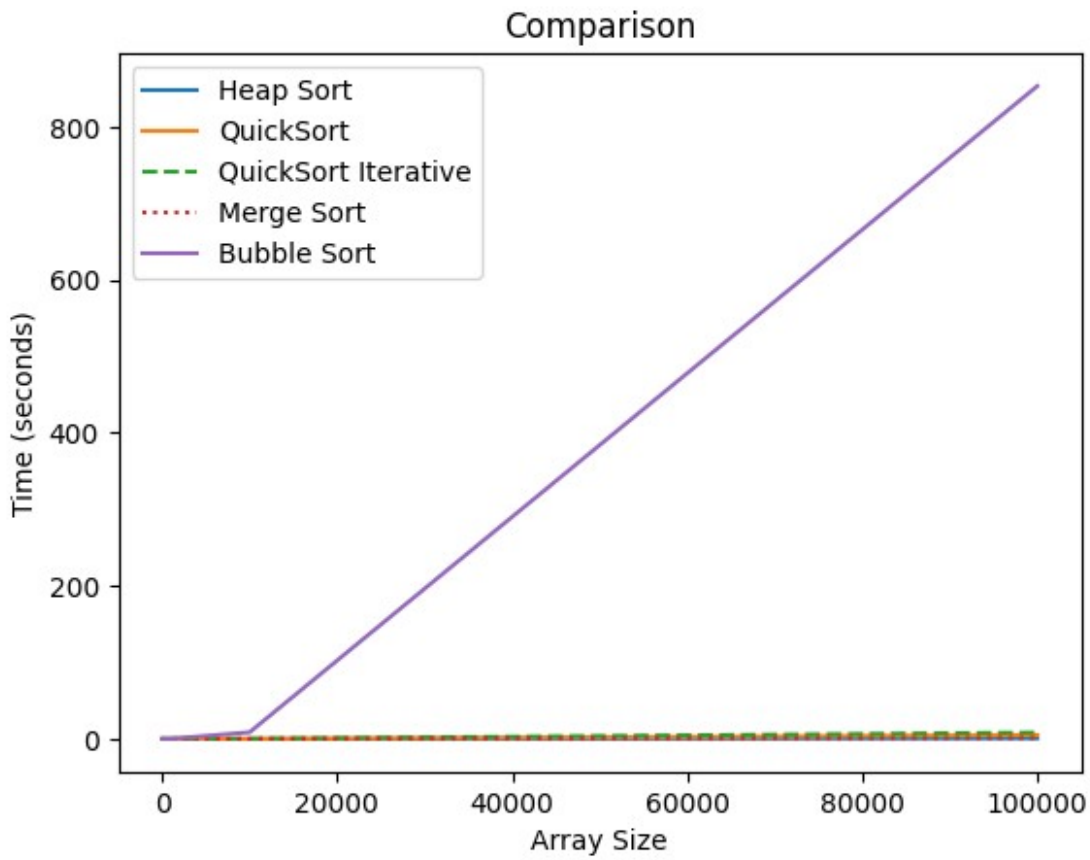
```
def bubble_sort(arr):
    n = len(arr)
    for i in range(n):
        # Flag to check if any swaps have been made
        swapped = False
        for j in range(0, n-i-1):
            if arr[j] > arr[j+1]:
                # Swap the elements
                arr[j], arr[j+1] = arr[j+1], arr[j]
                # Set the flag to true
                swapped = True
        # If no swaps were made, the array is sorted and we can stop
iterating
        if not swapped:
            break
    return arr

arr = []
times4 = [] #change
cases = []
j = 1
for k in range(5):
    j *= 10
    for i in range(j):
        arr.append(random.randint(0, 100))
    start = time.time()
    bubble_sort(arr)    #change
    end = time.time()
    times4.append(end-start) #change
    cases.append(j)

plt.plot(cases, times4,'bo-') #change
plt.title('Bubble Sort') #change
plt.xlabel('Array length')
plt.ylabel('Time (seconds)')
plt.show()
```




```
plt.plot(cases,times1, label="Heap Sort")
plt.plot(cases,times2, label="QuickSort")
plt.plot(cases,times5 ,'--', label="QuickSort Iterative")
plt.plot(cases,times3, ':', label="Merge Sort")
plt.title('Comparison (No Bubble Sort)')
plt.xlabel('Array Size')
plt.ylabel('Time (seconds)')
plt.legend()
plt.show()
```

```
plt.plot(cases,times1, label="Heap Sort")
plt.plot(cases,times2, label="QuickSort")
plt.plot(cases,times5 ,'--', label="QuickSort Iterative")
plt.plot(cases,times3, ':', label="Merge Sort")
plt.plot(cases,times4, '-', label="Bubble Sort")
plt.title('Comparison')
plt.xlabel('Array Size')
plt.ylabel('Time (seconds)')
plt.legend()
plt.show()
```


Comparison (No Bubble Sort)
Heap Sort
QuickSort
QuickSort Iterative
Merge Sort
Time (seconds)
0
2
4
6
8
0
20000
40000
60000
80000
100000
Array Size



Time/Size	10	100	1000	10000	100000
Heap Sort	0.0	0.000804901123046875	0.0062978267669677734	0.0608668327331543	0.7148110866546631
QuickSort	0.0011339187622070312	0.0	0.0020549297333276367	0.06781744956970215	4.936222553253174
QuickSort Iterative	0.0	0.0	0.0019943714141845703	0.09175348281860352	8.234138011932373
Merge Sort	0.0	0.0	0.0029916763305664062	0.037412166595458984	1.1898260116577148
Bubble Sort	0.0	0.0009970664978027344	0.0907900333404541	8.598121643066406	853.2158970832825