Name: Kayode Peter Temitope

Matric No: 208077

Department: Computer Science (200 Level)

CSC 236 Practical Project

Introduction to Algorithm

Search Algorithms

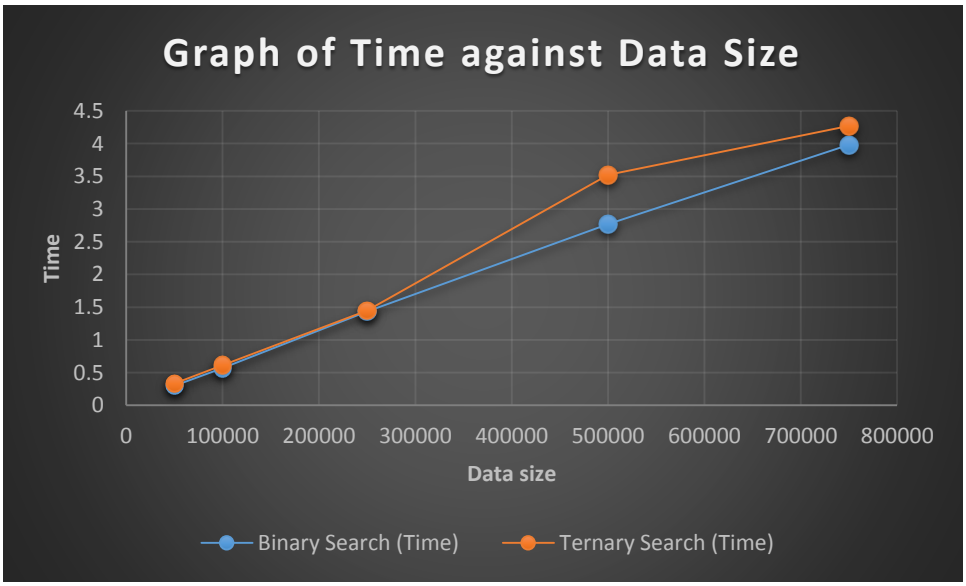
Unsorted Data

➢ *Search value is within the array*

Table of Values

S/N	Data Size	Search Value	Binary Search (Time)	Ternary Search (Time)
1	50000	70	0.2974	0.3359
2	100000	70	0.5703	0.6113
3	250000	70	1.4355	1.4506
4	500000	70	2.7709	3.5209
5	750000	70	3.9838	4.2709

Graph



Observation: I observed squarely in both binary and ternary search, that even though the search value (70) was in the array, the algorithm returned a feedback stating that the value was not found in the array.

Binary Search Python Code

```
import time
start = time.time()
import random
array = []
for i in range(0, 100000):
    n = random.randint(1, 100)
    array.append(n)
print(array)
#Using Binary Search to search for a value
def binarySearch(array, low, high, x):
    if high >= low:
        mid = (high + low)//2
        if array[mid] == x:
            return mid
        elif array[mid] > x:
            return binarySearch(array, low, mid-1, x)
        else:
            return binarySearch(array, mid+1, high, x)
    else:
        return -1
x = 120
result = binarySearch(array, 0, len(array)-1, x)
if result != -1:
    print("Element found at index:", str(result))
else:
    print("Element not found")
end = time.time()
print ("\nThe Runtime of the program is ",(end - start))
```

Ternary Search Python Code

```
import time
start = time.time()
import random
arr = []
for i in range(0, 750000):
    n = random.randint(1, 100)
    arr.append(n)
print(arr)
def ternarySearch(arr, x):
    left = 0
    right = len(arr) - 1
    while left <= right:
        mid1 = left + (right - left) // 3
        mid2 = left + 2 * (right - left) // 3
        if x == arr[left]:
            return left
        elif x == arr[right]:
            return right
        elif x < arr[left] or x > arr[right]:
            return -1
        elif x <= arr[mid1]:
            right = mid1
        elif x > arr[mid1] and x <= arr[mid2]:
            left = mid1 + 1
            right = mid2
        else:
            left = mid2 + 1
    return -1
def find(x):
    result = ternarySearch(arr, x)
    if result != -1:
```

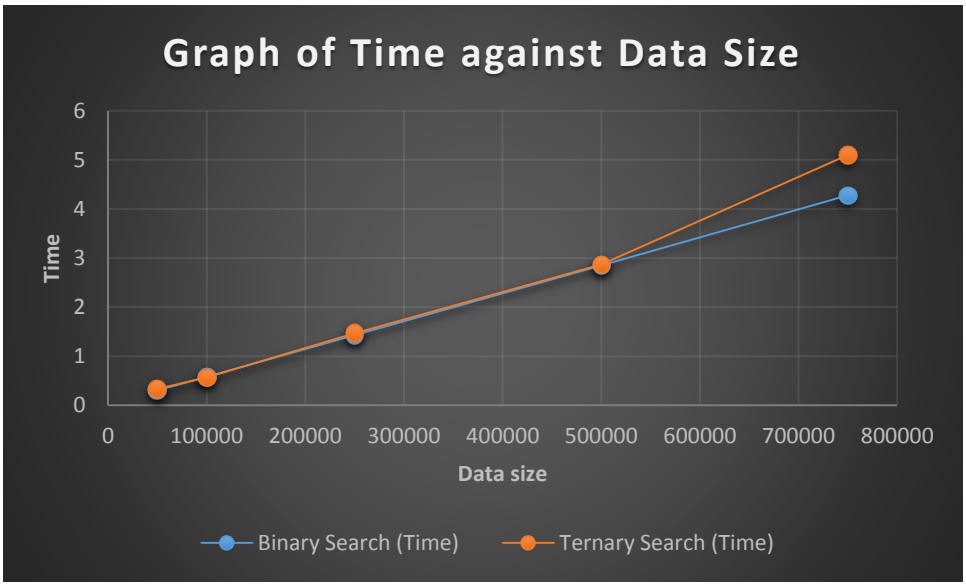
```
        print("The element", x, "is at the index", result)
    else:
        print("Element", x, "not found!")
find(70)
end = time.time()
print ("\nThe Runtime of the program is ",(end - start))
```

- *Search Value is not within the array*

Table of Values

S/N	Data Size	Search Value	Binary Search (Time)	Ternary Search (Time)
1	50000	120	0.2974	0.3320
2	100000	120	0.5830	0.5669
3	250000	120	1.4272	1.4663
4	500000	120	2.8509	2.8695
5	750000	120	4.2757	5.0970

Graph



Observation: Even though the search value was not in the array, it took almost the same time for when the search value was in the array before returning result.

Binary Search Python Code

```
import time
start = time.time()
import random
array = []
for i in range(0, 100000):
    n = random.randint(1, 100)
    array.append(n)
print(array)
#Using Binary Search to search for a value
def binarySearch(array, low, high, x):
    if high >= low:
        mid = (high + low)//2
        if array[mid] == x:
            return mid
        elif array[mid] > x:
            return binarySearch(array, low, mid-1, x)
        else:
            return binarySearch(array, mid+1, high, x)
    else:
        return -1
x = 120
result = binarySearch(array, 0, len(array)-1, x)
if result != -1:
    print("Element found at index:", str(result))
else:
    print("Element not found")
end = time.time()
print ("\nThe Runtime of the program is ",(end - start))
```

Ternary Search Python Code

```
import time
start = time.time()
import random
arr = []
for i in range(0, 750000):
    n = random.randint(1, 100)
    arr.append(n)
print(arr)
def ternarySearch(arr, x):
    left = 0
    right = len(arr) - 1
    while left <= right:
        mid1 = left + (right - left) // 3
        mid2 = left + 2 * (right - left) // 3
        if x == arr[left]:
            return left
        elif x == arr[right]:
            return right
        elif x < arr[left] or x > arr[right]:
            return -1
        elif x <= arr[mid1]:
            right = mid1
        elif x > arr[mid1] and x <= arr[mid2]:
            left = mid1 + 1
            right = mid2
        else:
            left = mid2 + 1
    return -1
def find(x):
    result = ternarySearch(arr, x)
    if result != -1:
```

```
        print("The element", x, "is at the index", result)
    else:
        print("Element", x, "not found!")
find(120)
end = time.time()
print ("\nThe Runtime of the program is ",(end - start))
```

Sorted Data

Sorting Technique used: Quick Sort

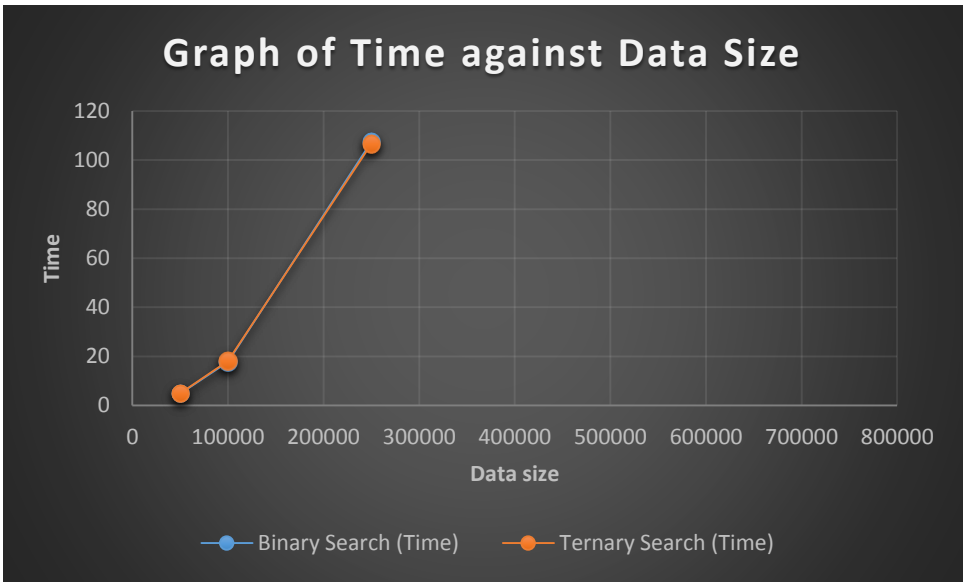
- *Search value is within the array*

Table of Values

S/N	Data Size	Search Value	Binary Search (Time)	Ternary Search (Time)
1	50000	70	4.8539	4.9672
2	100000	70	17.7802	18.1642
3	250000	70	107.5490	106.5317
4	500000	70	***	***
5	750000	70	***	***

****maximum recursion depth exceeded in comparison**; while trying to sort the data, the sorting algorithm(Quick Sort) reached its maximum recursion limit, thereby making it impossible for data size 500000 and 750000 to be sorted. Hence, the search for a value did not proceed.*

Graph



Observation: I observed that that the time taken for both the binary and ternary searching algorithm to complete for the same of number data size are relatively the same and they increase with data size correspondingly.

Binary Search Python Code

```
import time
start = time.time()
import random
array = []
for i in range(0, 50000):
    n = random.randint(1, 100)
    array.append(n)
#print(array)
#Sorting using Quick Sort
def quicksort(array, left, right):
    if left < right:
        partition_pos = partition(array, left, right)
        quicksort(array, left, partition_pos-1)
        quicksort(array, partition_pos+1, right)
        def partition(array, left, right):
    i = left
    j = right-1
    pivot = array[right]
    while i < j:
        while i < right and array[i] < pivot:
            i += 1
        while j > left and array[j] >= pivot:
            j -= 1
        if i < j:
            array[i], array[j] = array[j], array[i]
    if array[i] > pivot:
        array[i], array[right] = array[right], array[i]
    return i
quicksort(array, 0, len(array)-1)
print("\nThe sorted arrayay is:")
print(array)
```

```
#Using Binary Search to search for a value
def binarySearch(array, low, high, x):
    if high >= low:
        mid = (high + low)//2
        if array[mid] == x:
            return mid
        elif array[mid] > x:
            return binarySearch(array, low, mid-1, x)
        else:
            return binarySearch(array, mid+1, high, x)
    else:
        return -1
   x = 70
result = binarySearch(array, 0, len(array)-1, x)
if result != -1:
    print("Element found at index:", str(result))
else:
    print("Element not found")
end = time.time()
print ("\nThe Runtime of the program is ",(end - start))
```

Ternary Search Python Code

```
import time
start = time.time()
import random
arr = []
for i in range(0, 250000):
    n = random.randint(1, 100)
    arr.append(n)
#print(arr)
def quicksort(arr, left, right):
    if left < right:
        partition_pos = partition(arr, left, right)
        quicksort(arr, left, partition_pos-1)
        quicksort(arr, partition_pos+1, right)
def partition(arr, left, right):
    i = left
    j = right-1
    pivot = arr[right]
    while i < j:
        while i < right and arr[i] < pivot:
            i += 1
        while j > left and arr[j] >= pivot:
            j -= 1
        if i < j:
            arr[i], arr[j] = arr[j], arr[i]
    if arr[i] > pivot:
        arr[i], arr[right] = arr[right], arr[i]
    return i
quicksort(arr, 0, len(arr)-1)
print("\nThe sorted array is:")
print(arr)
#Seraching for a value
```

```
def ternarySearch(arr, x):
    left = 0
    right = len(arr) - 1
    while left <= right:
        mid1 = left + (right - left) // 3
        mid2 = left + 2 * (right - left) // 3
        if x == arr[left]:
            return left
        elif x == arr[right]:
            return right
        elif x < arr[left] or x > arr[right]:
            return -1
        elif x <= arr[mid1]:
            right = mid1
        elif x > arr[mid1] and x <= arr[mid2]:
            left = mid1 + 1
            right = mid2
        else:
            left = mid2 + 1
    return -1
def find(x):
    result = ternarySearch(arr, x)
    if result != -1:
        print("The element", x, "is at the index", result)
    else:
        print("Element", x, "not found!")
find(70)
end = time.time()
print ("\nThe Runtime of the program is ",(end - start))
```

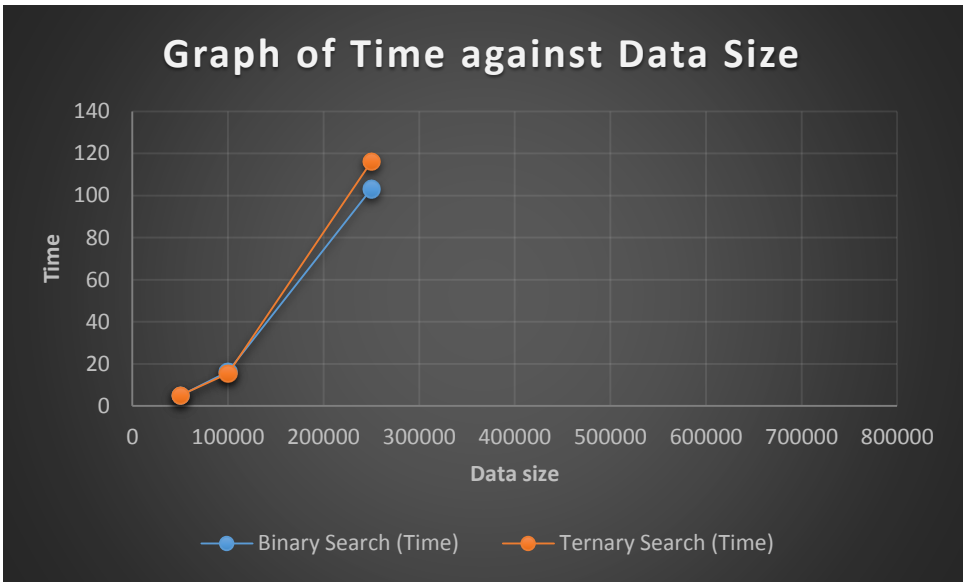
- *Search value is not within the array*

Table of Values

S/N	Data Size	Search Value	Binary Search (Time)	Ternary Search (Time)
1	50000	120	4.8528	5.0209
2	100000	120	16.1992	15.3008
3	250000	120	103.1360	116.1631
4	500000	120	***	***
5	750000	120	***	***

****maximum recursion depth exceeded in comparison**; while trying to sort the data, the sorting algorithm(Quick Sort) reached its maximum recursion limit, thereby making it impossible for data size 500000 and 750000 to be sorted. Hence, the search for a value did not proceed.*

Graph



Observation: Even while the data is sorted, the ternary searching algorithm still takes a bit more time to complete the searching than the binary search.

Binary Search Python Code

```
import time
start = time.time()
import random
array = []
for i in range(0, 50000):
    n = random.randint(1, 100)
    array.append(n)
#print(array)
#Sorting using Quick Sort
def quicksort(array, left, right):
    if left < right:
        partition_pos = partition(array, left, right)
        quicksort(array, left, partition_pos-1)
        quicksort(array, partition_pos+1, right)
        def partition(array, left, right):
    i = left
    j = right-1
    pivot = array[right]
    while i < j:
        while i < right and array[i] < pivot:
            i += 1
        while j > left and array[j] >= pivot:
            j -= 1
        if i < j:
            array[i], array[j] = array[j], array[i]
    if array[i] > pivot:
        array[i], array[right] = array[right], array[i]
    return i
quicksort(array, 0, len(array)-1)
print("\nThe sorted arrayay is:")
print(array)
```

```
#Using Binary Search to search for a value
def binarySearch(array, low, high, x):
    if high >= low:
        mid = (high + low)//2
        if array[mid] == x:
            return mid
        elif array[mid] > x:
            return binarySearch(array, low, mid-1, x)
        else:
            return binarySearch(array, mid+1, high, x)
    else:
        return -1
   x = 70
result = binarySearch(array, 0, len(array)-1, x)
if result != -1:
    print("Element found at index:", str(result))
else:
    print("Element not found")
end = time.time()
print ("\nThe Runtime of the program is ",(end - start))
```

Ternary Search Python Code

```
import time
start = time.time()
import random
arr = []
for i in range(0, 250000):
    n = random.randint(1, 100)
    arr.append(n)
#print(arr)
def quicksort(arr, left, right):
    if left < right:
        partition_pos = partition(arr, left, right)
        quicksort(arr, left, partition_pos-1)
        quicksort(arr, partition_pos+1, right)
def partition(arr, left, right):
    i = left
    j = right-1
    pivot = arr[right]
    while i < j:
        while i < right and arr[i] < pivot:
            i += 1
        while j > left and arr[j] >= pivot:
            j -= 1
        if i < j:
            arr[i], arr[j] = arr[j], arr[i]
    if arr[i] > pivot:
        arr[i], arr[right] = arr[right], arr[i]
    return i
quicksort(arr, 0, len(arr)-1)
print("\nThe sorted array is:")
print(arr)
#Seraching for a value
```

```
def ternarySearch(arr, x):
    left = 0
    right = len(arr) - 1
    while left <= right:
        mid1 = left + (right - left) // 3
        mid2 = left + 2 * (right - left) // 3
        if x == arr[left]:
            return left
        elif x == arr[right]:
            return right
        elif x < arr[left] or x > arr[right]:
            return -1
        elif x <= arr[mid1]:
            right = mid1
        elif x > arr[mid1] and x <= arr[mid2]:
            left = mid1 + 1
            right = mid2
        else:
            left = mid2 + 1
    return -1
def find(x):
    result = ternarySearch(arr, x)
    if result != -1:
        print("The element", x, "is at the index", result)
    else:
        print("Element", x, "not found!")
find(70)
end = time.time()
print ("\nThe Runtime of the program is ",(end - start))
```

General Observations

- **When Data is unsorted**
  - Generally, when the data is unsorted, the probability of getting the right outcome of a search in both binary and ternary search is almost zero. It was observed during the investigation that even though the search value was clearly in the array, the search algorithm returned an output stating that the value was not found in the array.
  - In binary search, the time taken for a search increases linearly with increase in the data size. In ternary search, although the time taken for a search also increases as the data size increases, but not linearly.
  - The ternary search algorithm takes more time to complete its process and return a result both when the search value is with the array and when the search value is not within the array.

- **When Data is sorted**
  - When there exist multiple occurrence of a search value, it returns the index of the first occurrence of the value in both binary and ternary search.
  - From the graph plotted, both the binary and the insertion search starts on closely the same value, but as the data size increases, the ternary search line deviates from the binary search line.
  - In both binary and ternary search, time taken for a search increases as the data size increases, but not linearly.

Inference

From the given observations and analysis, considering different data sizes with both sorted and unsorted array, it can be clearly seen that generally, the ternary search algorithm runs slower than the binary search algorithm. While the binary search algorithm runs faster than the ternary search algorithm

John's Hypothesis

John's Hypothesis is definitely wrong with the above observations and inference.
The ternary search algorithm does not run faster than the binary search. Instead, it runs slower than the binary search.