

# Binary Search

Binary Search List

1. left 0 right
2. left right
  - mid = (left + right) // 2
  - arr[mid] target
    - arr[mid] target mid
    - arr[mid] target right mid - 1
    - arr[mid] target left mid + 1
3. -1

- 
- $O(\log n)$  n
- 

## Example: Linear Search

```
def linear_search(list, key):  
    """If key is in the list returns its position in the list,  
    otherwise returns -1."""  
    for i, item in enumerate(list):  
        if item == key:  
            return i  
    return -1
```

## Example: Binary Search

```

def binary_search(list, key):
    """Returns the position of key in the list if found, -1 otherwise.

    List must be sorted.
    """

    # Sort the List
    list.sort()          # [ ] [ ] [ ] [ ]
    left, right = 0, len(list) - 1  # [ ] [ ] [ ] [ ] [ ] [ ]

    while left <= right:
        middle = (left + right) // 2  # [ ] [ ] [ ] [ ] [ ]

        if list[middle] == key:
            return middle          # [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ]
        if list[middle] > key:
            right = middle - 1     # [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ]
        if list[middle] < key:
            left = middle + 1      # [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ]
    return -1                    # [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] -1

# [ ]
my_list = [2, 4, 7, 12, 15, 21, 30, 34, 42]
target_number = 15

result = binary_search(my_list, target_number)

if result != -1:
    print(f"[ ] [ ] {target_number} [ ] [ ] [ ] [ ] [ ] [ ] {result}")
else:
    print(f"[ ] [ ] {target_number} [ ] [ ] [ ] [ ] [ ]")

```

## Example2: Binary Search

```

def find_item(list, item):
    #Returns True if the item is in the list, False if not.

    if len(list) == 0:
        return False

```

```
list.sort()

#Is the item in the center of the list?
middle = len(list)//2
if list[middle] == item:
    return True

#Is the item in the first half of the list?
if item < list[middle]:
    #Call the function with the first half of the list
    return find_item(list[:middle], item)
else:
    #Call the function with the second half of the list
    return find_item(list[middle+1:], item)

return False

list_of_names = ["Parker", "Drew", "Cameron", "Logan", "Alex", "Chris", "Terry", "Jamie", "Jordan", "Taylor"]

print(find_item(list_of_names, "Alex")) # True
print(find_item(list_of_names, "Andrew")) # False
print(find_item(list_of_names, "Drew")) # True
print(find_item(list_of_names, "Jared")) # False
```

1. ????? ???  
2. ???????? ??  
3. ??????? ???  
4. ???????? ???  
5. ?????? ??  
6. ?????? ??  
7. ?????? ??  
8. ??????? ??  
9. ?????? ??

```
def linear_search(list, key):  
    #Returns the number of steps to determine if key is in the list
```

```

#Initialize the counter of steps
steps=0
for i, item in enumerate(list):
    steps += 1
    if item == key:
        break
return steps

def binary_search(list, key):
    #Returns the number of steps to determine if key is in the list

    #List must be sorted:
    list.sort()

    #The Sort was 1 step, so initialize the counter of steps to 1
    steps=1

    left = 0
    right = len(list) - 1
    while left <= right:
        steps += 1
        middle = (left + right) // 2

        if list[middle] == key:
            break
        if list[middle] > key:
            right = middle - 1
        if list[middle] < key:
            left = middle + 1
    return steps

def best_search(list, key):
    steps_linear = linear_search(list, key)
    steps_binary = binary_search(list, key)
    results = "Linear: " + str(steps_linear) + " steps, "
    results += "Binary: " + str(steps_binary) + " steps. "
    if (steps_linear < steps_binary):
        results += "Best Search is Linear."
    elif (steps_linear > steps_binary):

```

```
        results += "Best Search is Binary."
    else:
        results += "Result is a Tie."

    return results

print(best_search([1, 2, 3, 4, 5, 6, 7, 8, 9, 10], 1))
#Should be: Linear: 1 steps, Binary: 4 steps. Best Search is Linear.

print(best_search([10, 2, 9, 1, 7, 5, 3, 4, 6, 8], 1))
#Should be: Linear: 4 steps, Binary: 4 steps. Result is a Tie.

print(best_search([10, 9, 8, 7, 6, 5, 4, 3, 2, 1], 7))
#Should be: Linear: 4 steps, Binary: 5 steps. Best Search is Linear.

print(best_search([1, 3, 5, 7, 9, 10, 2, 4, 6, 8], 10))
#Should be: Linear: 6 steps, Binary: 5 steps. Best Search is Binary.

print(best_search([5, 1, 8, 2, 4, 10, 7, 6, 3, 9], 11))
#Should be: Linear: 10 steps, Binary: 5 steps. Best Search is Binary.
```

---

Revision #18

Created 11 December 2024 15:35:13 by Admin

Updated 11 December 2024 17:48:10 by Admin