In [27]:
```
## Name : Shashank Dahake
## Roll No. : 59
## Practical No. : 1
## Aim: Analysis and comparison of brute force, insertion sort and Merge sort for the given scenarios.
```

In [28]:
```
import pandas as pd
import time as time
import matplotlib.pyplot as plt
```

In [29]:
```
# Statement-1: Consider the given file “data500.csv”. Write a program to read the file and
# find Minimum and Maximum using Insertion sort and a Divide and Conquer based Strastegy.
```

In [30]:
```
df = pd.read_csv('data500 - data500.csv')
nums = df['numbers'].tolist()
print(df)
```

```
     numbers
0        255
1         78
2        768
3        187
4        481
..       ...
495      405
496      180
497      763
498      978
499      294

[500 rows x 1 columns]
```

In [31]:
```
def insertion_sort(arr):
    for i in range(1, len(nums)):
        key = nums[i]
        j = i - 1
        while j >= 0 and key < nums[j]:
            nums[j + 1] = nums[j]
            j -= 1
    nums[j + 1] = key

arr1 = list(df.numbers)
start_time = time.perf_counter()
insertion_sort(arr1)
end_time = time.perf_counter()
execution_time = end_time - start_time
print("Max number:", nums[-1])
print("Min number:", nums[0])
print(f"Time taken by program is {execution_time} seconds.")
```

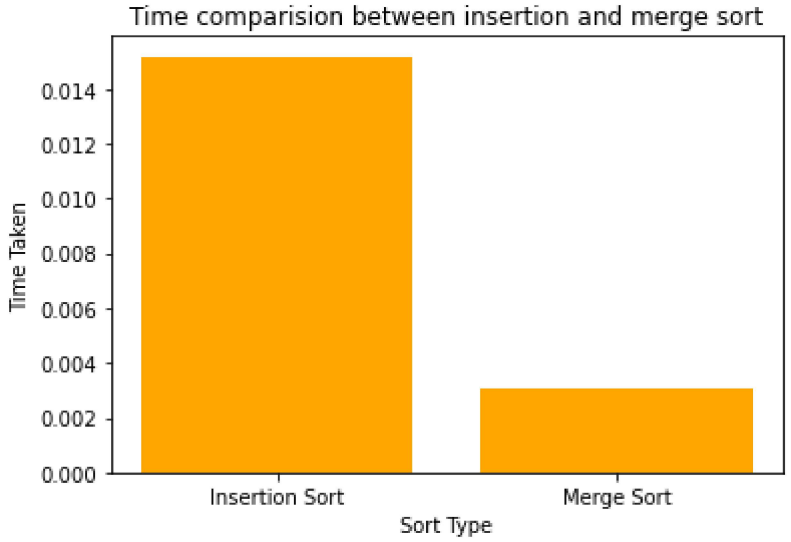
```
Max number: 998
Min number: 255
Time taken by program is 0.015177899971604347 seconds.
```

In [32]:
```
def merge(nums, left, mid, right):
    left_array = nums[left:mid + 1]
    right_array = nums[mid + 1:right + 1]
    i = j = 0
    k = left
    while i < len(left_array) and j < len(right_array):
        if left_array[i] <= right_array[j]:
            nums[k] = left_array[i]
            i += 1
        else:
            nums[k] = right_array[j]
            j += 1
        k += 1
    while i < len(left_array):
        nums[k] = left_array[i]
        i += 1
        k += 1
    while j < len(right_array):
        nums[k] = right_array[j]
        j += 1
        k += 1

def split(nums, left, right):
    if left < right:
        mid = (left + right) // 2
        split(nums, left, mid)
        split(nums, mid + 1, right)
        merge(nums, left, mid, right)

def merge_sort(arr):
    split(nums, 0, len(nums) - 1)

arr2 = list(df.numbers)
start_time = time.perf_counter()
merge_sort(arr2)
print("Max number:", nums[-1])
print("Min number:", nums[0])
end_time = time.perf_counter()
execution_time_merge = end_time - start_time
```

```
print(f"Time taken by program is {execution_time_merge} seconds.")
```

```
Max number: 998
Min number: 255
Time taken by program is 0.0030320000369101763 seconds.
```

In [33]:
```
sort_type = ["Insertion Sort", "Merge Sort"]
time = [execution_time, execution_time_merge]
plt.bar(sort_type, time, color = "orange")
plt.xlabel("Sort Type")
plt.ylabel("Time Taken")
plt.title("Time comparision between insertion and merge sort")
plt.show()
```



In [34]:
```
# Statement-2: In a school there is a class photograph. The class topper (say John) needs to
# stand in the middle and the other students need to stand height wise. Consider that there are N
# students in the class.
# Consider the inputs:
# N - Number of students in class excluding John.
# H - Height of Sam
# Array containing height of other students.

# Note:
# John should always stand in the middle and there&#39;s an equal number of students in his left and r
# Height of John is always unique.
```

In [25]:
```
# Approach:
# Sort Heights: Use merge sort to arrange the heights in ascending order.
# Find John's Position: Identify John's position in the sorted list and split the list into students s
# John.
# Balance Heights: Invite or cancel invitations randomly to balance the number of students taller and
# both sides are of equal length.
```

In [67]:
```
import random

def photograph(John_height, N, heights):
    heights.append(John_height)
    split(heights, 0 , N)

    for i in range(len(heights)):
        if heights[i] == John_height:
            left = heights[:i]
            right = heights[i + 1:]

    total_cost = 0
    operations = 0

    while len(left) > len(right):
        for i in range(abs(len(left) - len(right))):
            max_h = max(heights)
            h1 = random.randint(max_h + 1, max_h + 10)
            heights.append(h1)
            operations = operations + 1
            print(f"Operation: {operations} (We can invite a person with height {h1})")
            total_cost = total_cost + 1

    while len(left) < len(right) and heights:
        k = heights.pop()
        operations = operations + 1
        print(f"Operation: {operations} (We can cancel the invitation of student {k})")
        total_cost = total_cost + 1

    print("\nTotal Operations (Total_Cost): ", total_cost)
    print("The students are now arranged in sequence of heights: ")
    print(*heights)
    print(f"Where John's height is: {John_height}")

John_height = 170
N = 5
heights = [160, 165, 175, 180]
photograph(John_height, N, heights)
```

```
Total Operations (Total_Cost):  0
The students are now arranged in sequence of heights:
160 165 170 175 180
Where John's height is: 170
```