Identify the time and memory complexity of each of the functions

```
//initialization
int partition (int arr[], int low, int high)
{
    int pivot = arr[high];
    int i = (low - 1);
    for (int j = low; j <= high- 1; j++)
    {
    if (arr[j] <= pivot)
    {
    i++;
    swap(&arr[i], &arr[j]);
    }
}
swap(&arr[i + 1], &arr[high]);
return (i + 1);
}
```

Memory complexity in this code is equal : number of high.

```
//Quick sort Function
void quickSort(int arr[], int low, int high)
{
    if (low < high)
    {
    int pi = partition(arr, low, high);
    quickSort(arr, low, pi - 1);
    quickSort(arr, pi + 1, high);
    }
}
```

Memory complexity in this code is equal : 1

```
//Print Element Of Array
void printArray_Quick(int arr[], int size) {
    int i;
    for (i=0; i < size; i++)
    printf("This is Quick Sort : %d \n", arr[i]);

}
```

Memory complexity in this code is equal : number of size.

```
            // Insertion Sort Function
//Insertion Sort Function
void insertionSort(int arr[], int n)
{
    int i, key, j;
    for (i = 1; i < n; i++) {
        key = arr[i];
        j = i - 1;

        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = key;

    }

}
```

Memory complexity in this code is equal : n

```c
// Print Function
void printArray_Insertion(int arr[], int n)
{
    int i;
    for (i = 0; i < n; i++)
    printf("\nThis is Insertion Sort: %d \n", arr[i]);


}
```

Memory complexity in this code is equal : n