Assignment 3
Devam Desai
IIT2022035

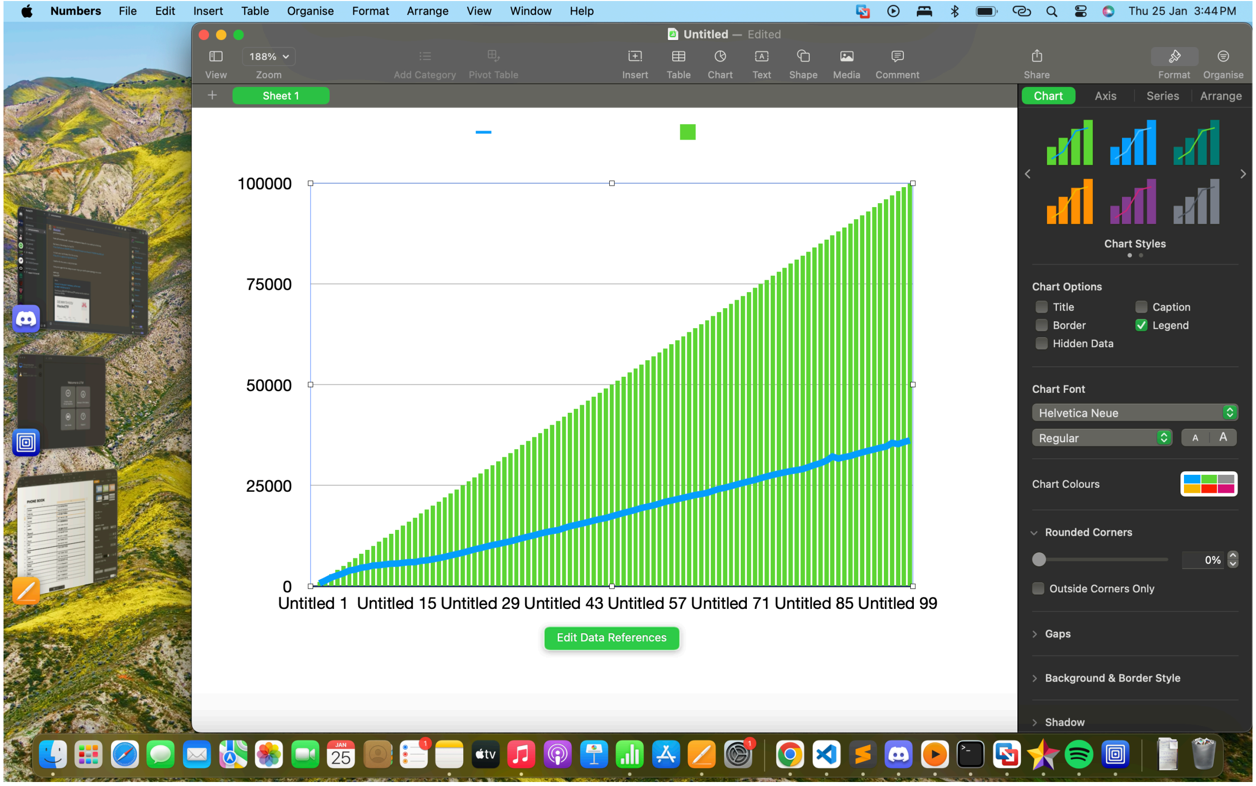
Code used for analysis:

```
vector<int> arr;
    int maxi = 1e5;
    for (int i = 0;i < n;i++){
        arr.push_back(rand()%(maxi));
    }
    auto start = high_resolution_clock::now();

    kthSmallest(arr,0,n-1,100);
    auto stop = high_resolution_clock::now();
    auto duration = duration_cast<microseconds>(stop - start);
    cout << (duration).count() << endl;
```

Q1. Mergesort

Graph of time vs n for mergesort. In mergesort k will not make a difference. So graph will be a constant line.
YAxis : Time
X axis: n

```
void merge(vint &array, int const left, int const mid,
           int const right)
{
    int const subArrayOne = mid - left + 1;
    int const subArrayTwo = right - mid;

    // Create temp arrays
    vint leftArray(subArrayOne);
    vint rightArray(subArrayTwo);

    // Copy data to temp arrays leftArray[] and rightArray[]
    for (auto i = 0; i < subArrayOne; i++)
        leftArray[i] = array[left + i];
    for (auto j = 0; j < subArrayTwo; j++)
        rightArray[j] = array[mid + 1 + j];

    auto indexOfSubArrayOne = 0, indexOfSubArrayTwo = 0;
    int indexOfMergedArray = left;
    // Merge the temp arrays back into array[left..right]
    while (indexOfSubArrayOne < subArrayOne
           && indexOfSubArrayTwo < subArrayTwo) {
        if (leftArray[indexOfSubArrayOne]
            <= rightArray[indexOfSubArrayTwo]) {
            array[indexOfMergedArray]
                = leftArray[indexOfSubArrayOne];
            indexOfSubArrayOne++;
        }
        else {
            array[indexOfMergedArray]
                = rightArray[indexOfSubArrayTwo];
            indexOfSubArrayTwo++;
        }
        indexOfMergedArray++;
    }

    // Copy the remaining elements of
    // left[], if there are any
    while (indexOfSubArrayOne < subArrayOne) {
        array[indexOfMergedArray]
            = leftArray[indexOfSubArrayOne];
        indexOfSubArrayOne++;
        indexOfMergedArray++;
    }

    // Copy the remaining elements of
    // right[], if there are any
    while (indexOfSubArrayTwo < subArrayTwo) {
        array[indexOfMergedArray]
            = rightArray[indexOfSubArrayTwo];
        indexOfSubArrayTwo++;
        indexOfMergedArray++;
    }
}




// begin is for left index and end is right index
// of the sub-array of arr to be sorted
void mergeSort(vint &array, int const begin, int const end)
{
    if (begin >= end)
        return;
    int mid = begin + (end - begin) / 2;
    mergeSort(array, begin, mid);
    mergeSort(array, mid + 1, end);
    merge(array, begin, mid, end);
}
```



Zoom: 188%

Chart Options

☐ Title ☐ Caption
☐ Border ☑ Legend
☐ Hidden Data

Chart Font: Helvetica Neue

Regular

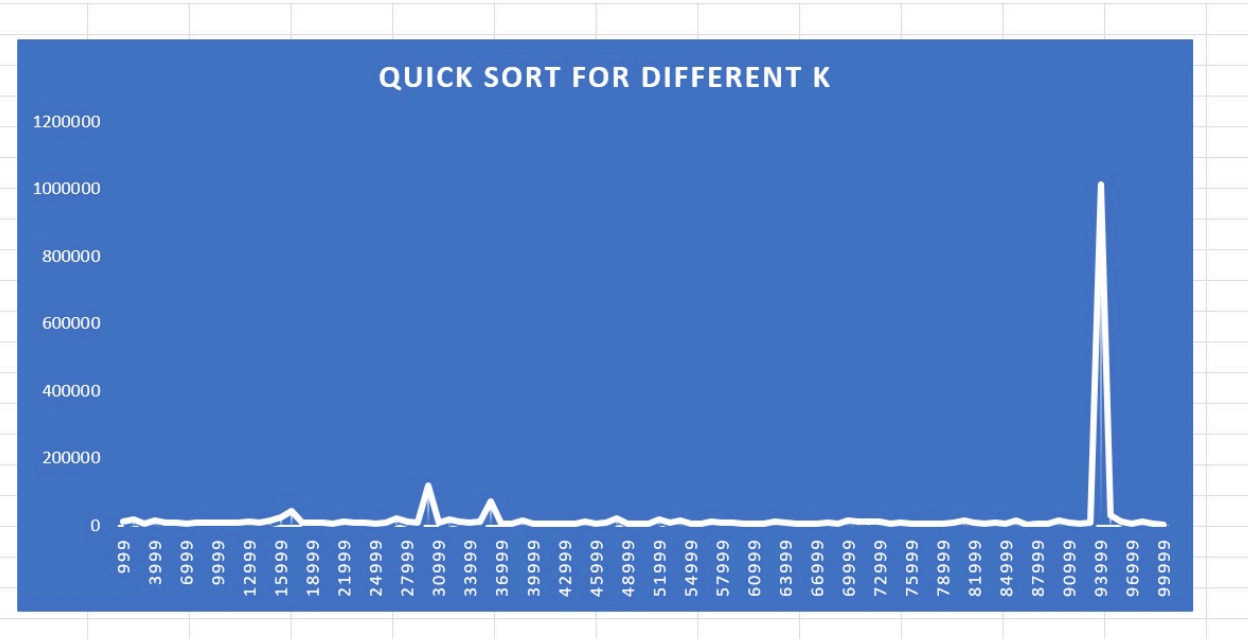
Rounded Corners: 0%

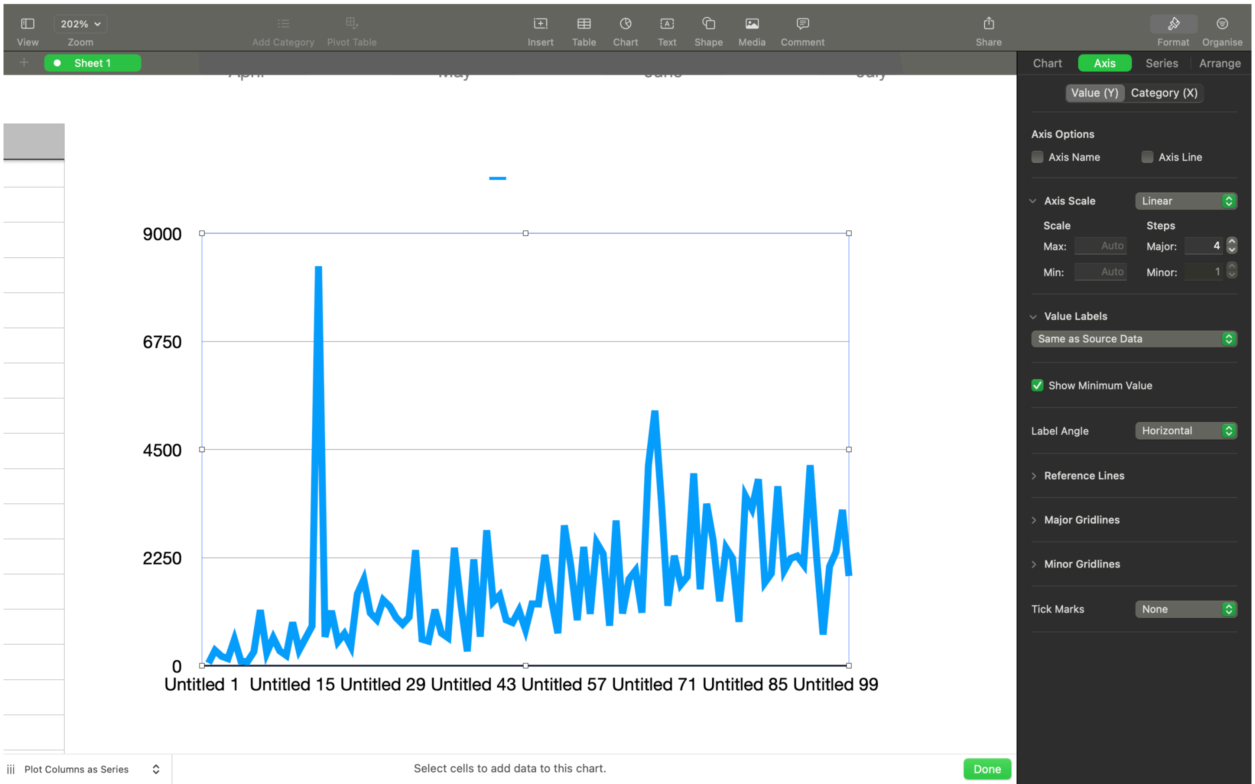
☐ Outside Corners Only

Q2: Quicksort(Using quickselect to find kth minimum)
Graph of time vs n ;

X axis- n
Y axis - time

```
int kthSmallest(vector<int> arr, int l, int r, int K)
{
    // If k is smaller than number of elements in array
    if (K > 0 && K <= r - l + 1) {

        // Partition the array around last element and get
        // position of pivot element in sorted array
        int pos = partition(arr, l, r);

        // If position is same as k
        if (pos - l == K - 1)
            return arr[pos];
        if (pos - l > K - 1) // If position is more, recur
                             // for left subarray
            return kthSmallest(arr, l, pos - 1, K);

        // Else recur for right subarray
        return kthSmallest(arr, pos + 1, r,
                           K - pos + l - 1);
    }

    // If k is more than number of elements in array
    return INT_MAX;
}

void swap(vector<int>::iterator a, vector<int>::iterator b)
{
    int temp = *a;
    *a = *b;
    *b = temp;
}

// Standard partition process of QuickSort(). It considers
// the last element as pivot and moves all smaller element
// to left of it and greater elements to right
int partition(vector<int> arr, int l, int r)
{
    int x = arr[r], i = l;
    for (int j = l; j <= r - 1; j++) {
        if (arr[j] <= x) {
            swap(arr.begin()+i, arr.begin()+j);
            i++;
        }
    }

    swap(arr.begin()+i, arr.begin()+r);
    return i;
}
```





Axis Options

□ Axis Name □ Axis Line

Axis Scale: Linear

Scale — Max: Auto, Min: Auto

Steps — Major: 4, Minor: 1

Value Labels: Same as Source Data

☑ Show Minimum Value

Label Angle: Horizontal

Reference Lines

Major Gridlines

Minor Gridlines

Tick Marks: None

Select cells to add data to this chart.