ADS Homework 4

Joshua Morris

4. 1

a) Implement merge + insertion sort ✓

```
// prototyped functions
void merge(int arr[], int left, int middle, int right);
void insertionSort(int arr[], int front, int end);
void mergeSort(int arr[], int front, int end, int k);
void randomizeArr(int arr[], int size);
void arrFill(int arr[], int size);
```

Prototype functions to be used in main

```
int main(){

    // delete files in string array before append data
    string files[3] = {"avgCase.csv", "bestCase.csv", "worstCase.csv"};
    char fileNameChar[15];

    for (int fileIndex = 0; fileIndex < 3; fileIndex++){
        // copying over the string into the char array & removing
        strcpy(fileNameChar, files[fileIndex].c_str());
        remove(fileNameChar);
    }

    ofstream outFile;
    // setting sample size to 3000 & init array with sampleSize
    int sampleSize = 3000;
    int arr[sampleSize];
```

Clyde through files

Effectively make sub dir

If files exist delete them.

Set sample size and set that array size

```
// loop for different k values
for(int k = 0; k < sampleSize; k+=15){

    // ------------------------------------------------------------
    // best case

    // open file in append mode
    outFile.open("bestCase.csv", ios_base::app);

    // get array of elements
    arrFill(arr, sampleSize);

    // get current time
    auto startCaseBest = high_resolution_clock::now();

    // run merge sort (array passed, front index, end index, k value)
    mergeSort(arr, 0, sampleSize, k);

    //stop time by getting current time and subtract to find total time
    auto stopCaseBest = high_resolution_clock::now();
    auto caseBest = duration_cast<microseconds>(stopCaseBest - startCaseBest);

    outFile << k << "," << (double)caseBest.count() << endl;

    outFile.close();
```

Use variable k to see how time complexity changes

Open CSV file in append mode

function to fill whole array

Start timer

Run merge sort get execution time

close file

Write data to CSV

```
// ------------------------------------------------------------------------
// average case

// open file in append mode
outFile.open("avgCase.csv", ios_base::app);

// get random array to be sorted
randomizeArr(arr, k);

// get current time
auto startCaseAvg = high_resolution_clock::now();

// run merge sort
mergeSort(arr, 0, sampleSize, k);

//stop time by getting current time and subtract to find total time
auto stopCaseAvg = high_resolution_clock::now();
auto caseAvg = duration_cast<microseconds>(stopCaseAvg - startCaseAvg);

outFile << k << "," << (double)caseAvg.count() << endl ;

outFile.close();
```

randomize array get average case

Start clock

Merge

Stop clock

write to file

```
    // ------------------------------------------------------------------
    //Worst case

    // open file in append mode
    outFile.open("worstCase.csv", ios_base::app);

    // get ordered array
    arrFill(arr, k);

    // sort in decending order
    sort(arr, arr + k, greater<int>());

    // get current time
    auto startCaseWorst = high_resolution_clock::now();

    // run merge sort
    mergeSort(arr, 0, sampleSize, k);

    //stop time by getting current time and subtract to find total time
    auto stopCaseWorst = high_resolution_clock::now();
    auto caseWorst = duration_cast<microseconds>(stopCaseWorst - startCaseWorst);

    outFile << k << "," << (double)caseWorst.count() << endl;

    outFile.close();
}

return 0;
```

fill array with with elements

Sort elements in decreasing fashion

Sort elements

```java
/*
much help with understanding and implementing merge sort,
https://www.geeksforgeeks.org/merge-sort/
*/
void merge(int arr[], int left, int middle, int right){

    // get sizes to use for arrays
    int size1 = middle - left + 1;
    int size2 = right - middle;

    // Create temp arrays
    int tempArrLeft[size1];
    int tempArrRight[size2];

    // move data into the temp arrays
    for(int i = 0; i < size1; i++){
        tempArrLeft[i] = arr[left+i];
    }
    for(int j = 0; j < size2; j++){
        tempArrRight[j] = arr[middle+1+j];
    }

    // Initial index of first subarray
    int index1 = 0;
    // Initial index of second subarray
    int index2 = 0;
    // Initial index of merged subarray
    int index3 = left;
```

Sizes for 2 arrays

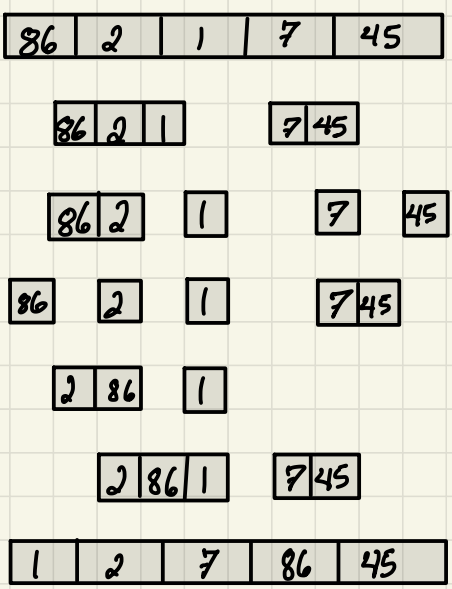
Split single array into 2 to be sorted.

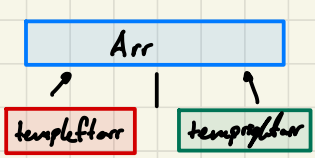
Arr to left

Arr to right

```java
    // cycle through arrays sorting elements
    while(index1 < size1 && index2 < size2){
        if(tempArrLeft[index1] <= tempArrRight[index2]){

            arr[index3] = tempArrLeft[index1];
            index1++;
        }else{
            arr[index3] = tempArrRight[index2];
            index2++;
        }
        index3++;
    }

    // copy data over back into main arr from right and left arrays
    while(index1 < size1){
        arr[index3] = tempArrLeft[index1];
        index1++;
        index3++;
    }
    while(index2 < size2){
        arr[index3] = tempArrRight[index2];
        index2++;
        index3++;
    }
}
```

Merge sort visual feature by example





```
/*
much help with understanding and implementing merge sort,
https://www.geeksforgeeks.org/merge-sort/
*/
void mergeSort(int arr[], int front, int end, int k){

    /*
    when mergesort reaches sequences of k (k is variable,
    thus changes in main), we run insertion sort.
    */
    if((end-front) <= k){
        insertionSort(arr, front, end);

    // else continue merge sort
    }else{
        // calculate middle value
        int middle = front+(end-front)/2;

        // call mergeSort function on front to middle part array
        mergeSort(arr, front, middle, k);
        // call mergeSort function on middle to end part array
        mergeSort(arr, middle+1, end, k);
        // call merge function
        merge(arr, front, middle, end);
    }
}
```

Depending on k value then mergesort do insertion sort

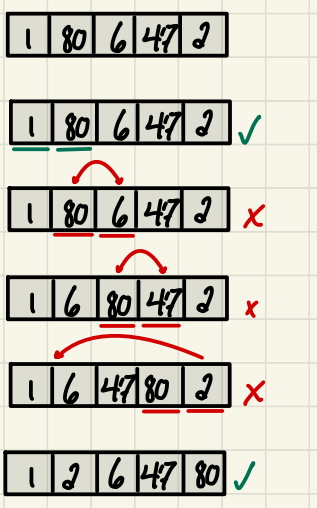
Middle calculate

Recursive function on the two halves found

Call merge to combine elements

```
// help with insertion sort from https://www.geeksforgeeks.org/insertion-sort/
void insertionSort(int arr[], int front, int end){
    int i, key, j;

    /*
        start at the first element to the end element (as only want
        to sort these elements in the larger array). If wanted to sort all
        elements start at index 0.
    */
    for(i = front; i < end; i++){

        j = i;
        key = arr[i];
        j = i-1;

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

        arr[j+1] = key;
    }
}
```

Insertion Sort by example



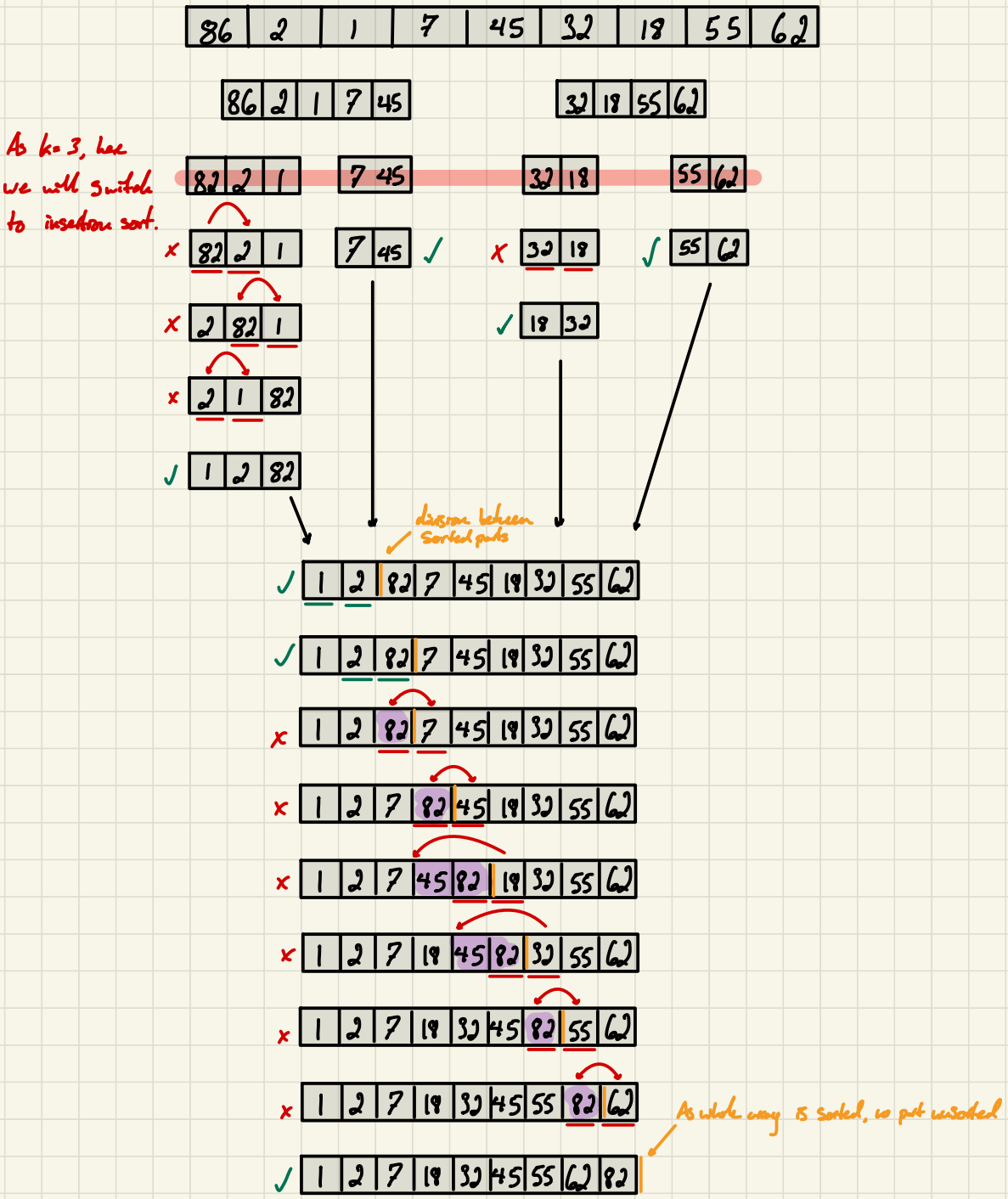

```
// function to fill array with random values
void randomizeArr(int arr[], int size){
    // initialize random
    srand(static_cast <unsigned int>(time(0)));

    for (int i = 0; i < size; i++){
        arr[i] = rand() % 100 + 1;
    }
}

// function to fill array with index times 2
void arrFill(int arr[], int size){
    for (int i = 0; i < size; i++){
            arr[i] = i*2;
    }
}
```

Initialize rand

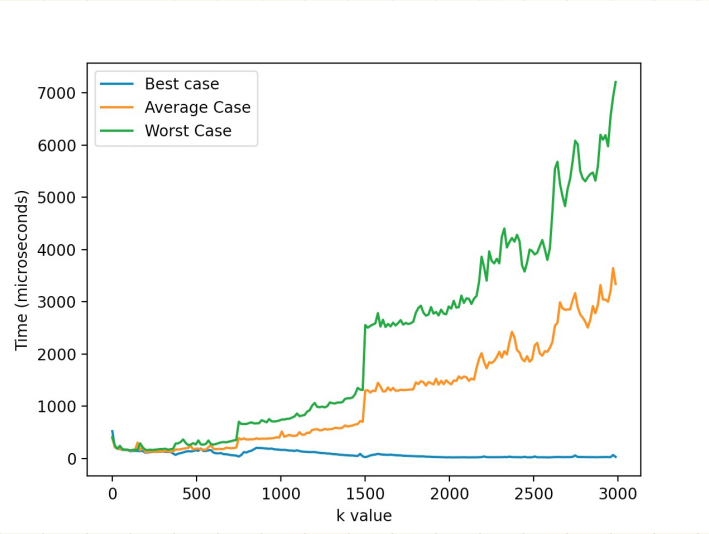
Random values

Fill the array with elements

As the question asked, insertion sort, was combined with merge sort. This would like as, as follows, with k = 3.



b) Plot best, average, and worst case



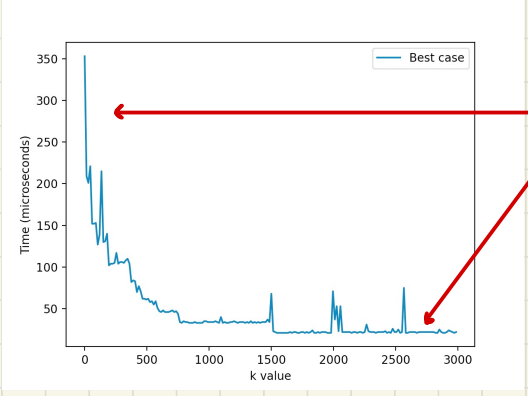
From data collected using the C++ program above.

C)

The time complexities for the three cases, followed as such,

Best Case

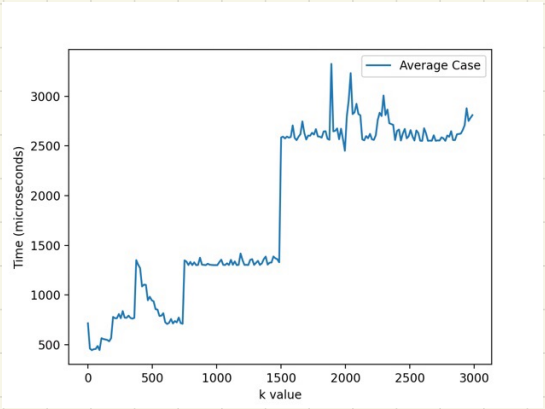
From looking at the graph, the best case varies very little as increased k values are applied. When speaking to the time complexities of our algorithm, we found that k is greatly smaller. This plays the key role in finding the time complexity, as when k gets larger we insertion sort more.



As seen here, when k gets larger, the time complexity decreases.

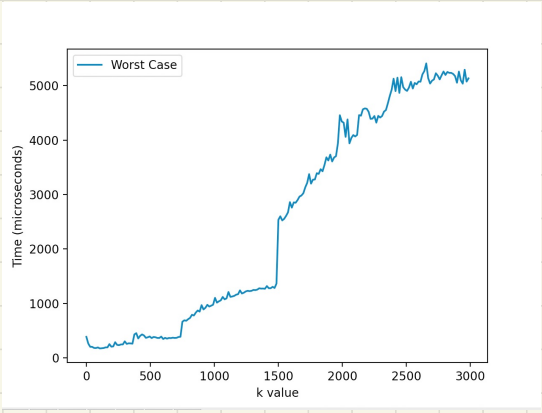
Average Case

The average case can be seen to get a greater time complexity once k increases. Merge sort has the average time complexity of $\Theta(n \log n)$ while insertion sort has $\Theta(n^2)$. As we would take the higher time complexity, we say that the average case would be $\Theta(n^2)$.



Worst Case

While the k value increases, so does the time complexity. As insertion sort has the time complexity of $\Theta(n^2)$ for the worst case and merge sort $\Theta(n \log n)$, we can say that the algorithm has the time complexity of $\Theta(n^2)$.



D)

While keeping the time complexities of merge sort and insertion sort, we know the table,

	Insertion Sort	Merge Sort
Best case	$\Theta(n)$	$\Theta(n \lg n)$
Average case	$\Theta(n^2)$	$\Theta(n \lg n)$
Worst case	$\Theta(n^2)$	$\Theta(n \lg n)$

Thus, we can thank of them to use which algorithms and then which would be more useful when keeping in mind time complexity. If we are given them to sort that are mostly pre-sorted, then insertion sort would be preferable as it has the best time complexity of $\Theta(n)$. This rather than $\Theta(n \log n)$, mergesort. Hence a larger k would be good.

If the items on the list are less sorted than more or randomly chosen, then merge sort would be preferred as insertion sort for average and worst have the time complexity of $\Theta(n^2)$. This, going with $\Theta(n \log n)$, for which would do better with a lower k value.

4.2

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

a)

$$T(n) = 36T\left(\frac{n}{6}\right) + 2n$$

$a = 36$

$n = n$

$b = 6$

$$\Theta(n^{\log_b a}) \rightarrow \Theta(n^{\log_6 36}) \rightarrow \Theta(n^2)$$

$$\Theta(n^2) > 2n$$

with $T(n) = n^2$ and $f(n) = n$, we notice that $f(n)$ is polynomially smaller than $T(n)$. Thus, it falls into case 1 of the master theorem with the time complexity being $\Theta(n^2)$.

b)

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

$$5T\left(\frac{n}{3}\right) + 18n^{1.2}$$

$a = 5$

$b = 3$

$n = n$

Not polynomially greater or less, so approx.

$$\Theta(n^{\log_b a}) \rightarrow \Theta(n^{\log_3 5}) \rightarrow \Theta(n^{\log_3 5}) \approx 1.2$$ thus the time complexity is $\Theta(n^{\log_3 6} \cdot \log(1.2))$

C)

$T(n) = 12T(\frac{n}{2}) + n^2 \log(n)$

↓

$T(n) = aT(\frac{n}{b}) + \Theta(n^k \log^p n)$ ⟶ Answer formulea

$T(\Theta) = \Theta(n^{\log_b a} \cdot \log^{p+1} n)$

Fill in

$a = 12$
$b = 2$
$k = 2$
$p = 1$

$\Theta(n^{\log_2 12} \cdot \log^2(n))$

D)

$T(n) = 3T(\frac{n}{5}) + T(\frac{n}{2}) + 2^n$

Case 1

$T(n) = 4T(\frac{n}{5}) + 2^n$

$n^{\log_5 4} < 2^n$

Hence, $T(n) = 4T(\frac{n}{5}) + 2^n \to \Theta(2^n)$

Case 3

$T(n) = 4T(\frac{n}{2}) + 2^n$

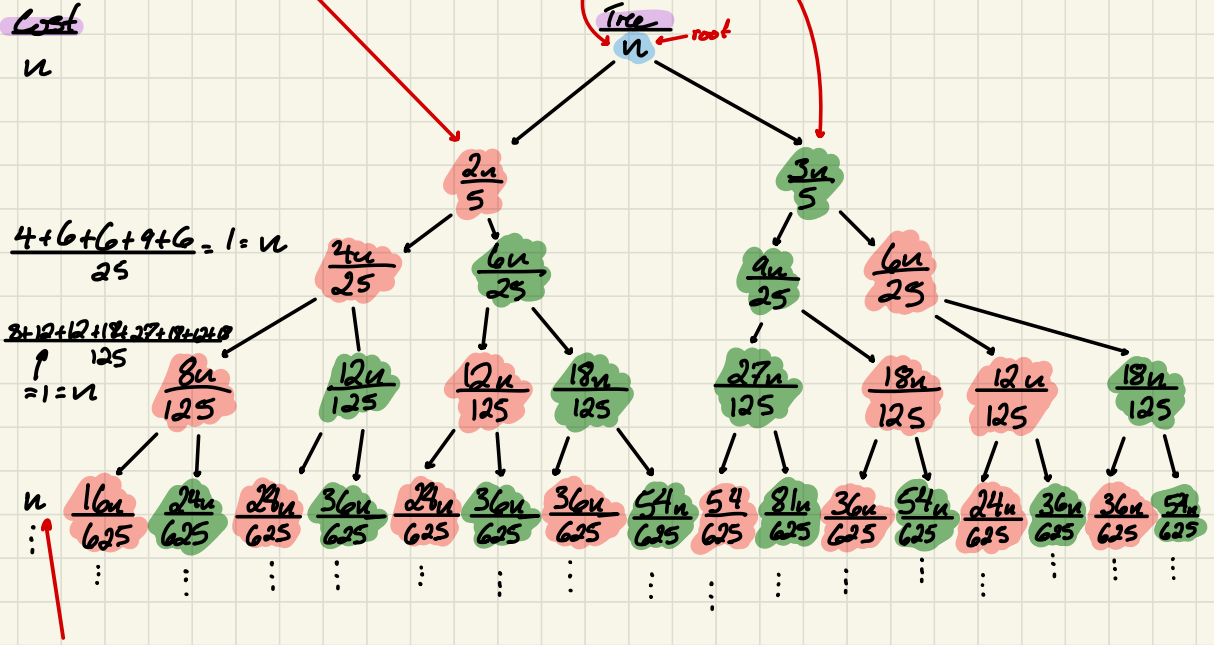
$n^{\log_2 2^1} = 2^n$

Hence, $T(n) = 4T(\frac{n}{2}) + 2^n \to \Theta(n^2)$

Therefore, $T(n) = 3T(\frac{n}{5}) + T(\frac{n}{2}) + 2^n \to \Theta(n^2)$

E) $T(n) = T(\frac{2n}{5}) + T(\frac{3n}{5}) + \Theta(n)$

Help in understanding from,
https://youtu.be/0D2-sYen23E?t=151



$$\frac{16+24+24+36+24+36+36+54+54+81+36+54+24+36+36+54}{625} = 1 = n$$

total cost = (cost of leaf nodes · total leaf nodes) + (Sum of cost at each level of internal nodes)

total cost = $L_c + I_c$

n

Hence,

$n + n\log_{\frac{5}{3}}(n)$

As n is not as great as $n\log(n)$ ~~n~~ $+ n\log_{\frac{5}{3}}(n)$

$n\log_{5/3}(n)$

From cost

$n \cdot k$ ← number of rows

$n\log_{\frac{5}{3}}(n)$

As $\frac{3n}{5}$ has a greater time complexity wise it

$\frac{2n}{5}$ or $\frac{3n}{5}$

$T(\frac{2n}{5^k}) = T(1)$

$2n = 5^k$

$n = \frac{5}{2}^k$

$k = \log_{\frac{5}{2}}(n)$

$T(\frac{3n}{5^k}) = T(1)$

$3n = 5^k$

$n = \frac{5}{3}^k$

$k = \log_{\frac{5}{3}}(n)$