Holly Robertson
Comp 311
Homework 7
March 16, 2019

Problem 1 [8 points]

Show the execution of quick sort on the following array. You should show the array after each call to partition. Explicitly state your pivot selection strategy.

34 19 3 22 5 2 46 30

I picked the median pivot strategy. After researching this sorting algorithm, the median strategy on average gives back the most useful number to pivot off of. Below is the algorithm used to set the pivot to the median average between the first, middle and last element. **Code Posted at GitHub (didn't want to take too much room here)**

```
getPivot(left, right)
left = 0
right = 7 (length – 1)
center = (left + right)/2
          = 3
```

34	19	3	22	5	2	46	30

```
if (a[left] > a[center])
          swap(left,center)
```

22	19	3	34	5	2	46	30

```
if (a[left] > a[right])
          swap(left, right)
```

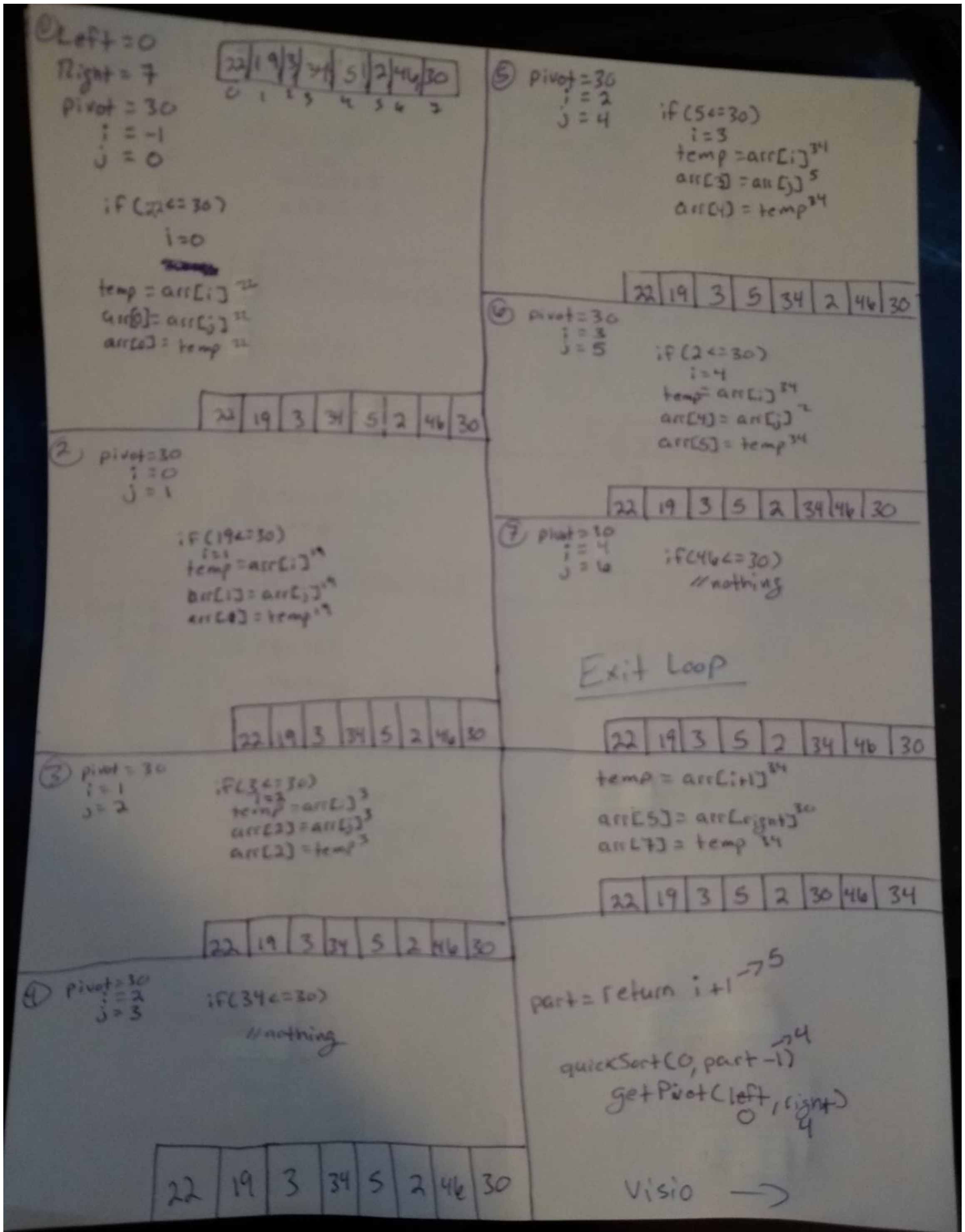
22	19	3	34	5	2	46	30

```
if (a[center] > a[right])
          swap(center, right)
```

22	19	3	30	5	2	46	34

```
swap(center, right)
```

22	19	3	34	34	2	46	30



① Left = 0
Right = 7
pivot = 30
i = -1
j = 0

22	19	3	34	5	2	46	30
0	1	2	3	4	5	6	7

if (22 <= 30)
i = 0
temp = arr[i] ^{22}
arr[0] = arr[j] ^{22}
arr[0] = temp ^{22}

22	19	3	34	5	2	46	30

② pivot = 30
i = 0
j = 1

if (19 <= 30)
i = 1
temp = arr[i] ^{19}
arr[1] = arr[j] ^{19}
arr[0] = temp ^{19}

22	19	3	34	5	2	46	30

③ pivot = 30
i = 1
j = 2

if (3 <= 30)
i = 2
temp = arr[i] ^{3}
arr[2] = arr[j] ^{3}
arr[2] = temp ^{3}

22	19	3	34	5	2	46	30

④ pivot = 30
i = 2
j = 3

if (34 <= 30)
//nothing

22	19	3	34	5	2	46	30

⑤ pivot = 30
i = 2
j = 4

if (5 <= 30)
i = 3
temp = arr[i] ^{34}
arr[3] = arr[j] ^{5}
arr[4] = temp ^{34}

22	19	3	5	34	2	46	30

⑥ pivot = 30
i = 3
j = 5

if (2 <= 30)
i = 4
temp = arr[i] ^{34}
arr[4] = arr[j] ^{2}
arr[5] = temp ^{34}

22	19	3	5	2	34	46	30

⑦ pivot = 30
i = 4
j = 6

if (46 <= 30)
//nothing

Exit Loop

22	19	3	5	2	34	46	30

temp = arr[i+1] ^{34}

arr[5] = arr[right] ^{30}
arr[7] = temp ^{34}

22	19	3	5	2	30	46	34

part = return i+1 $\rightarrow 5$

quickSort(0, part-1 $\rightarrow 4$)
getPivot($\underset{0}{\text{left}}$, $\underset{4}{\text{right}}$)

Visio $\longrightarrow$

```
getPivot(left, right)
left = 0
right = 4 (partition – 1)
center = (left + right)/2
          = 2
```

★		★		★			
22	19	3	5	2	30	46	34

```
if (a[left] > a[center])
          swap(left,center)
```

3	19	22	5	2	30	46	34

```
if (a[left] > a[right])
          swap(left, right)
```

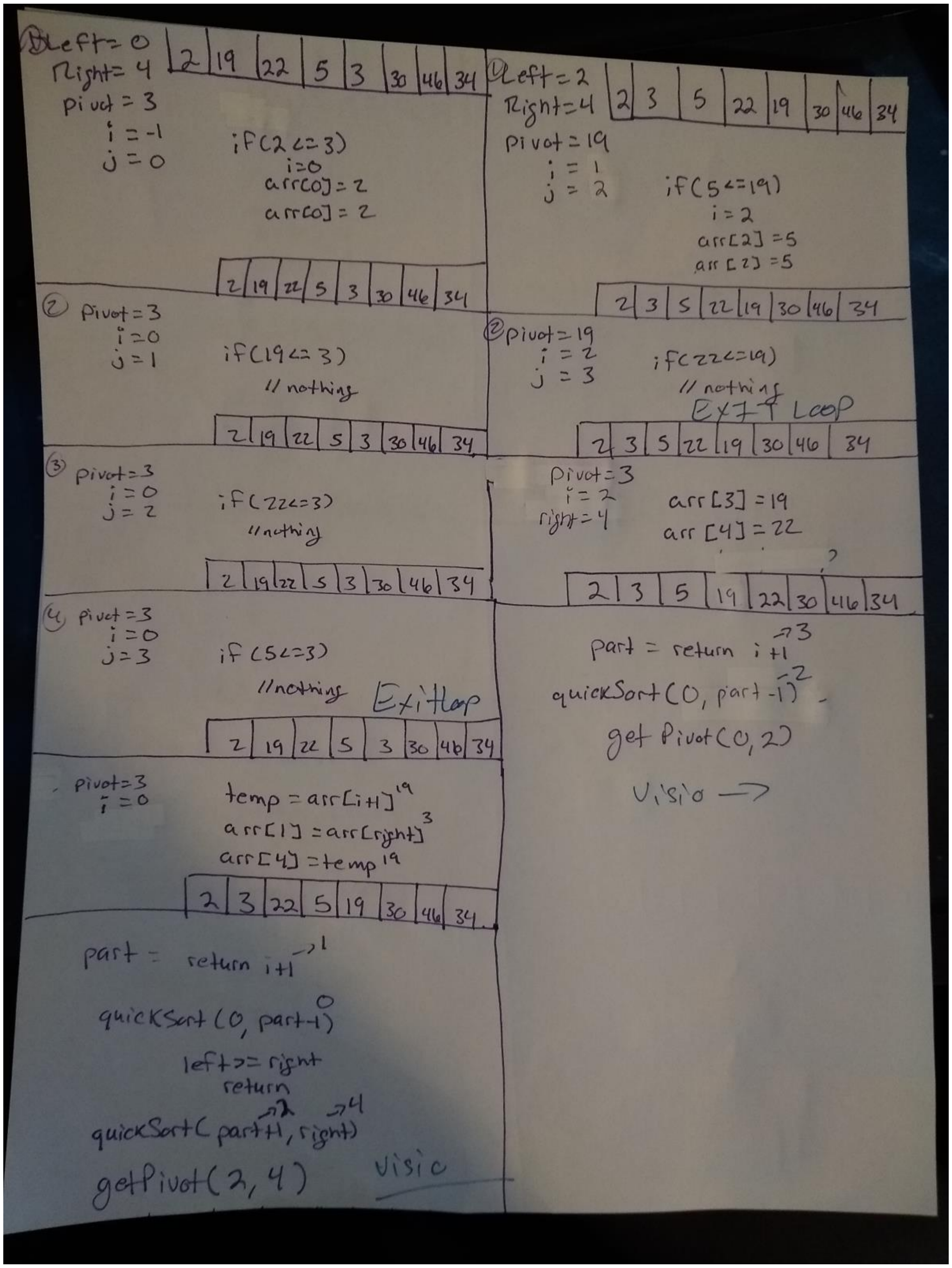
2	19	22	5	3	30	46	34

```
if (a[center] > a[right])
          swap(center, right)
```

2	19	3	5	22	30	46	34

```
swap(center, right)
```

2	19	22	5	3	30	46	34



① Left = 0
Right = 4
pivot = 3
i = -1
j = 0

2	19	22	5	3	30	46	34

```
if(2 <= 3)
    i=0
    arr[0] = 2
    arr[0] = 2
```

2	19	22	5	3	30	46	34

② pivot = 3
i = 0
j = 1

```
if(19 <= 3)
    // nothing
```

2	19	22	5	3	30	46	34

③ pivot = 3
i = 0
j = 2

```
if(22 <= 3)
    //nothing
```

2	19	22	5	3	30	46	34

④ pivot = 3
i = 0
j = 3

```
if(5 <= 3)
    //nothing    Exit loop
```

2	19	22	5	3	30	46	34

pivot = 3
i = 0

```
temp = arr[i+1]  19
arr[1] = arr[right]  3
arr[4] = temp  19
```

2	3	22	5	19	30	46	34

```
part = return i+1   →1

quickSort(0, part-1)   0
    left >= right
    return

quickSort(part+1, right)   →2  →4

getPivot(2, 4)      visio
```

① Left = 2
Right = 4

2	3	5	22	19	30	46	34

pivot = 19
i = 1
j = 2

```
if(5 <= 19)
    i = 2
    arr[2] = 5
    arr[2] = 5
```

2	3	5	22	19	30	46	34

② pivot = 19
i = 2
j = 3

```
if(22 <= 19)
    // nothing
    Exit Loop
```

2	3	5	22	19	30	46	34

pivot = 3
i = 2
right = 4

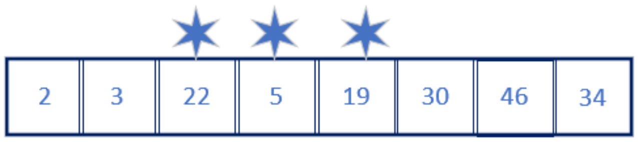
```
arr[3] = 19
arr[4] = 22
```

?

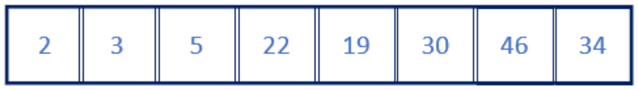
2	3	5	19	22	30	46	34

```
part = return i+1   →3

quickSort(0, part-1)   →2

get Pivot(0, 2)

Visio →
```

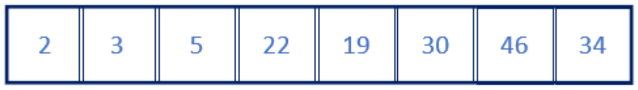
```
getPivot(left, right)
left = 2 (partition + 1)
right = 4
center = (left + right)/2
       = 3
```



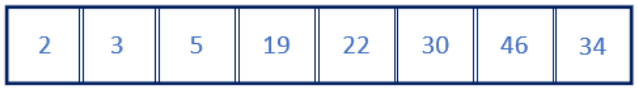
```
if (a[left] > a[center])
        swap(left,center)
```



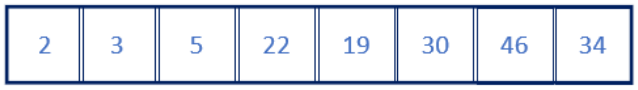
```
if (a[left] > a[right])
        swap(left, right)
```



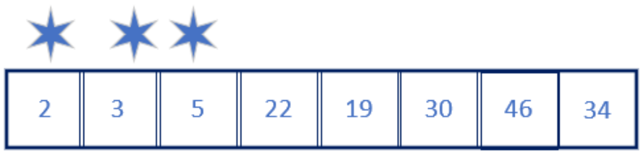
```
if (a[center] > a[right])
        swap(center, right)
```



```
swap(center, right)
```



```
getPivot(left, right)
left = 0
right = 2 (partition - 1)
center = (left + right)/2
       = 1
```

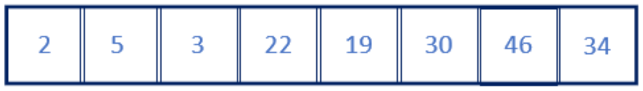


```
if (a[left] > a[center])
        swap(left,center)
```

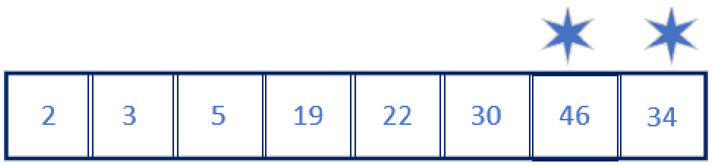
```
if (a[left] > a[right])
        swap(left, right)
```

```
if (a[center] > a[right])
        swap(center, right)
```

```
swap(center, right)
```

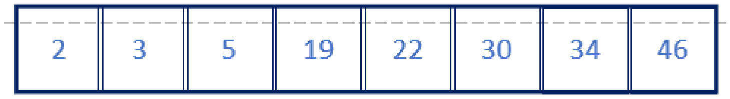


```
getPivot(left, right)
left = 6 (partition + 1)
right = 7
center = (left + right)/2
         = 6
```



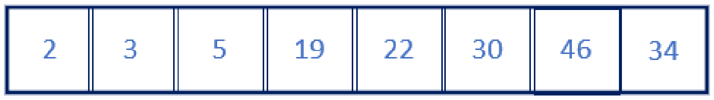
```
if (a[left] > a[center])
         swap(left,center)
```

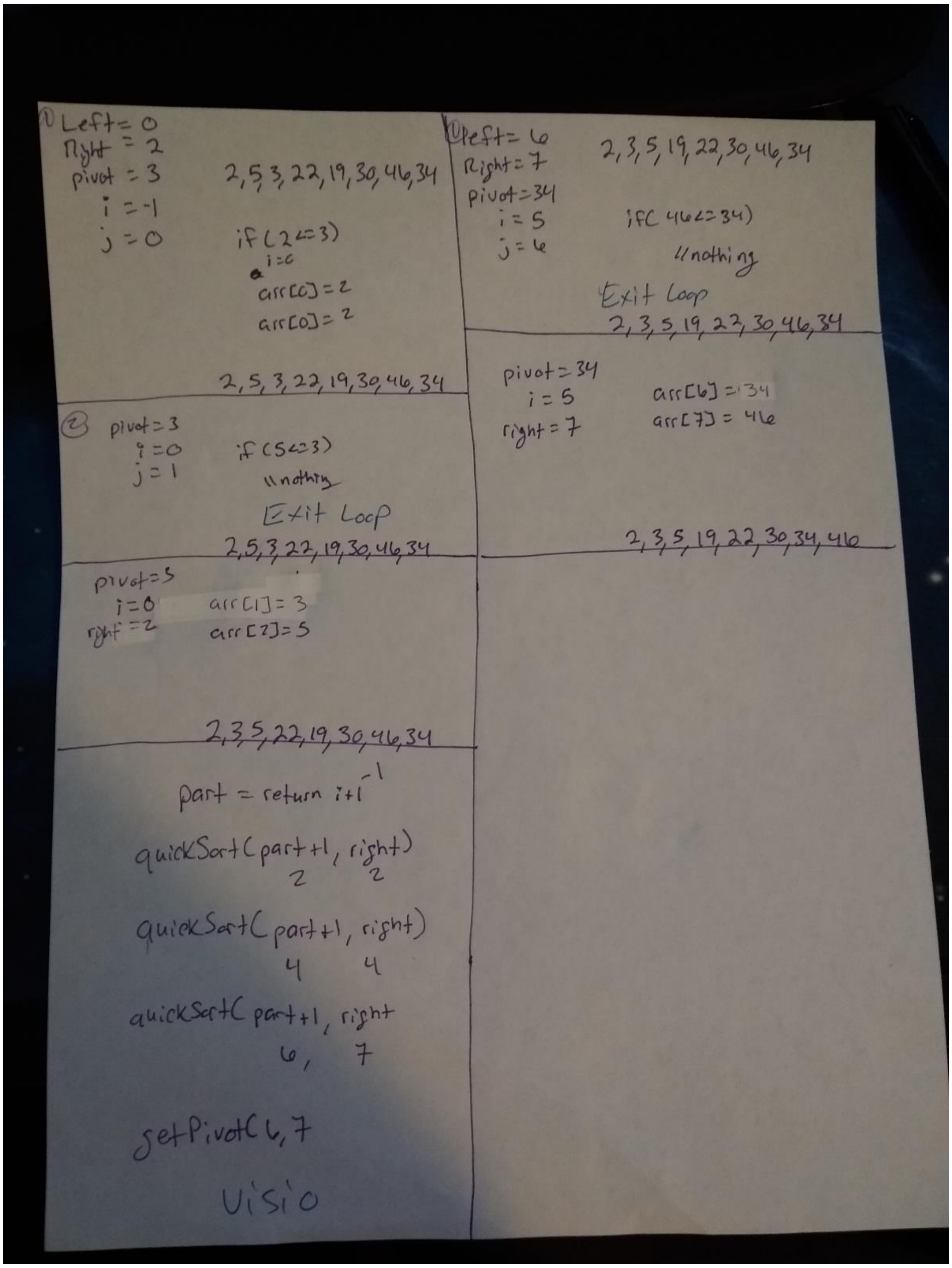
```
if (a[left] > a[right])
         swap(left, right)
```



```
if (a[center] > a[right])
         swap(center, right)
```

```
swap(center, right)
```





① Left = 0
right = 2
pivot = 3
i = -1
j = 0

2,5,3,22,19,30,46,34

if (2<=3)
i=0
arr[0] = 2
arr[0] = 2

2,5,3,22,19,30,46,34

② pivot = 3
i = 0
j = 1

if (5<=3)
//nothing
Exit Loop

2,5,3,22,19,30,46,34

pivot = 3
i = 0
right = 2

arr[1] = 3
arr[2] = 5

2,3,5,22,19,30,46,34

part = return $i+1^{-1}$

quickSort(part+1, right)
2 2

quickSort(part+1, right)
4 4

quickSort(part+1, right
6, 7

setPivot(6,7

Visio

① Left = 6
Right = 7
pivot = 34
i = 5
j = 6

2,3,5,19,22,30,46,34

if(46<=34)
//nothing
Exit Loop

2,3,5,19,22,30,46,34

pivot = 34
i = 5
right = 7

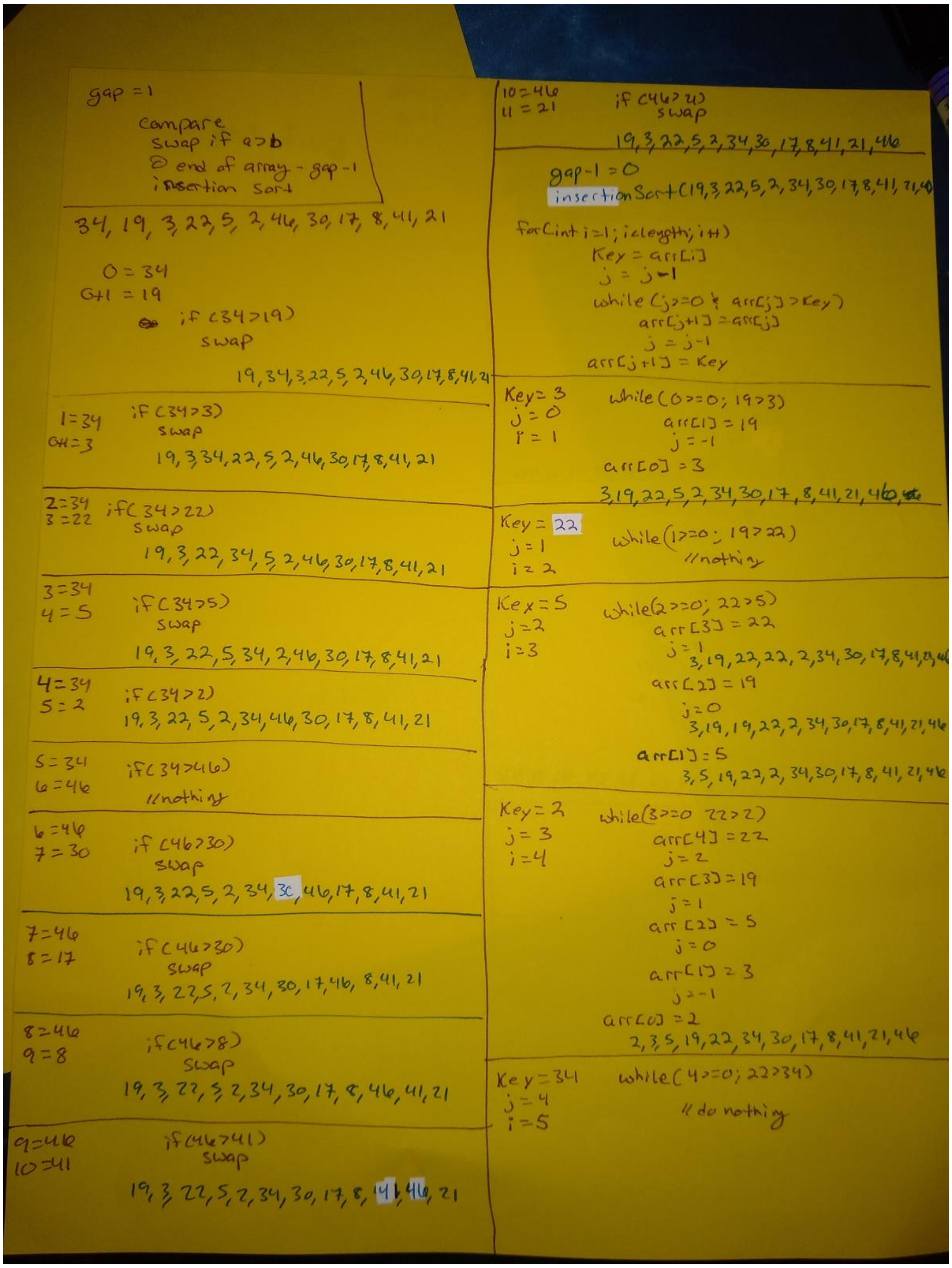
arr[6] = 34
arr[7] = 46

2,3,5,19,22,30,34,46

Problem 2 [8 points]

Show the execution of shell sort on the following array. Use gaps of three, two, and one. You should show the array after each pass.

```
34 19 3 22 5 2 46 30 17 8 41 23
```



```
gap =1

     compare
      swap if a>b
      0 end of array - gap -1
      insertion sort
```

34, 19, 3, 22, 5, 2, 46, 30, 17, 8, 41, 21

```
 0 = 34
G+1 = 19
      if (34>19)
         swap

            19,34,3,22,5,2,46,30,17,8,41,21
```

```
1=34     if (34>3)
G+1=3       swap
         19,3,34,22,5,2,46,30,17,8,41,21
```

```
2=34  if(34>22)
3=22     swap
         19,3,22,34,5,2,46,30,17,8,41,21
```

```
3=34
4=5      if(34>5)
            swap
         19,3,22,5,34,2,46,30,17,8,41,21
```

```
4=34
5=2      if(34>2)
         19,3,22,5,2,34,46,30,17,8,41,21
```

```
5=34     if(34>46)
6=46       //nothing
```

```
6=46
7=30     if (46>30)
            swap
         19,3,22,5,2,34,30,46,17,8,41,21
```

```
7=46
8=17     if(46>30)
            swap
         19,3,22,5,2,34,30,17,46,8,41,21
```

```
8=46
9=8      if(46>8)
            swap
         19,3,22,5,2,34,30,17,8,46,41,21
```

```
9=46      if(46>41)
10=41        swap

         19,3,22,5,2,34,30,17,8,41,46,21
```

```
10=46    if (46>21)
11=21       swap

         19,3,22,5,2,34,30,17,8,41,21,46
```

```
gap-1 =0
insectionSort(19,3,22,5,2,34,30,17,8,41,21,46)

for(int i=1; i<length; i++)
      Key = arr[i]
       j = j-1
      while (j>=0 & arr[j]>Key)
           arr[j+1] = arr[j]
            j = j-1
      arr[j+1] = Key
```

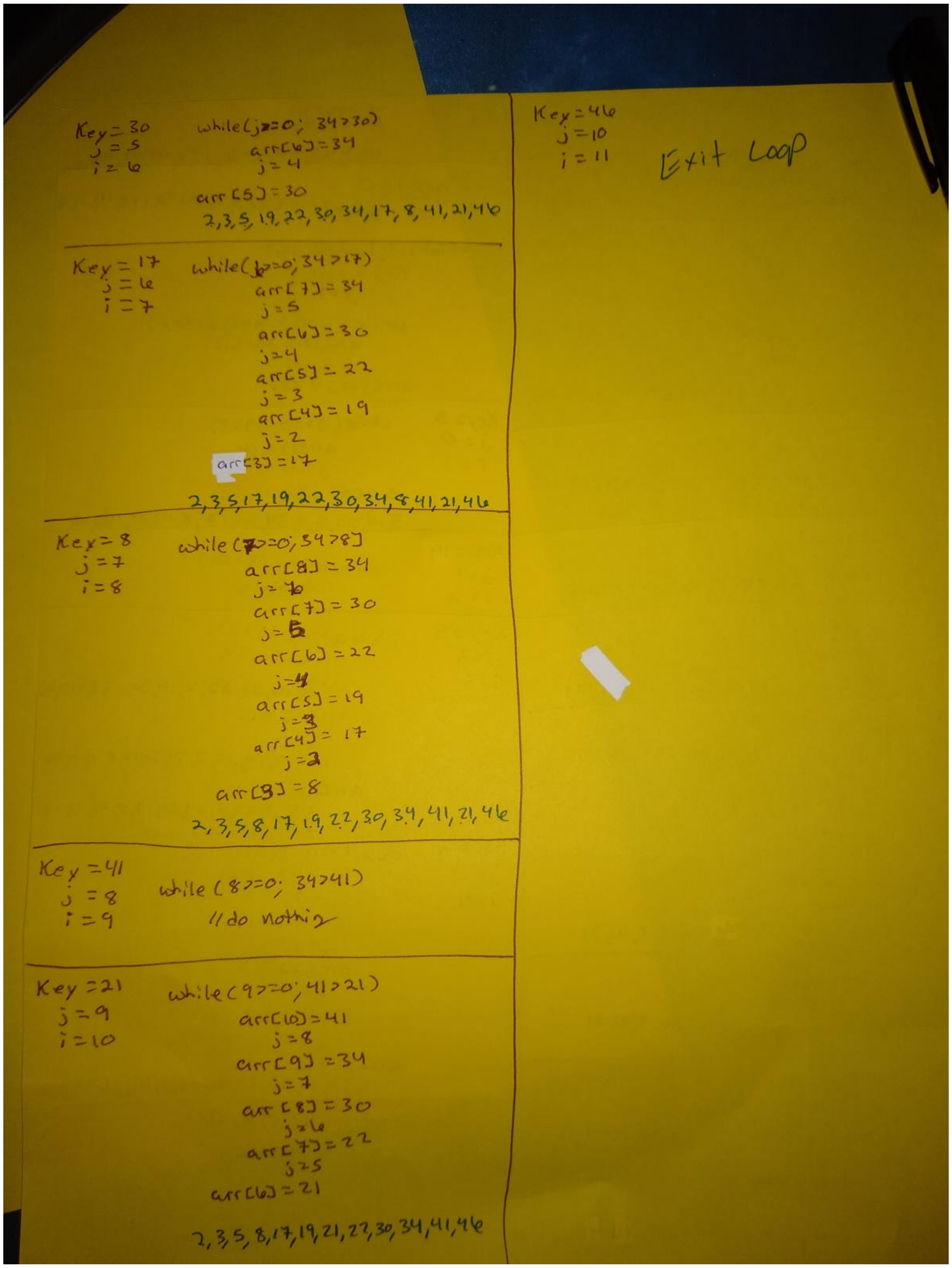
```
Key= 3    while(0>=0; 19>3)
 j=0          arr[1] = 19
 i=1          j = -1
          arr[0] = 3
          3,19,22,5,2,34,30,17,8,41,21,46,
```

```
Key= 22   while(1>=0; 19>22)
 j=1           //nothing
 i=2
```

```
Kex=5    while(2>=0; 22>5)
 j=2         arr[3] = 22
 i=3          j=1
               3,19,22,22,2,34,30,17,8,41,21,46
             arr[2] = 19
              j=0
               3,19,19,22,2,34,30,17,8,41,21,46
           arr[1]=5
               3,5,19,22,2,34,30,17,8,41,21,46
```

```
Key=2    while(3>=0 22>2)
 j=3         arr[4] =22
 i=4          j=2
             arr[3]=19
              j=1
             arr[2] = 5
              j=0
             arr[1] = 3
              j=-1
         arr[0] =2
           2,3,5,19,22,34,30,17,8,41,21,46
```

```
Key=34    while(4>=0; 22>34)
 j=4
 i=5               // do nothing
```



```
Key = 30     while(j>=0; 34>30)
 j = 5              arr[6] = 34
 i = 6              j = 4
             arr[5] = 30
             2,3,5,19,22,30,34,17,8,41,21,46
-----------------------------------------------
Key = 17     while(6>=0; 34>17)
 j = 6              arr[7] = 34
 i = 7              j = 5
                    arr[6] = 30
                    j = 4
                    arr[5] = 22
                    j = 3
                    arr[4] = 19
                    j = 2
             arr[3] = 17
             2,3,5,17,19,22,30,34,8,41,21,46
-----------------------------------------------
Key = 8      while(7>=0; 34>8]
 j = 7              arr[8] = 34
 i = 8              j = 6
                    arr[7] = 30
                    j = 5
                    arr[6] = 22
                    j = 4
                    arr[5] = 19
                    j = 3
                    arr[4] = 17
                    j = 2
             arr[3] = 8
             2,3,5,8,17,19,22,30,34,41,21,46
-----------------------------------------------
Key = 41
 j = 8       while(8>=0; 34>41)
 i = 9              //do nothing
-----------------------------------------------
Key = 21     while(9>=0; 41>21)
 j = 9              arr[10] = 41
 i = 10             j = 8
                    arr[9] = 34
                    j = 7
                    arr[8] = 30
                    j = 6
                    arr[7] = 22
                    j = 5
             arr[6] = 21

             2,3,5,8,17,19,21,22,30,34,41,46
```

```
Key = 46
 j = 10
 i = 11      Exit Loop
```

GAP = 2

Pass												
1	34	19	3	22	5	2	46	30	17	8	41	21
2	3	19	34	22	5	2	46	30	17	8	41	21
3	3	19	34	22	5	2	46	30	17	8	41	21
4	3	19	5	22	34	2	46	30	17	8	41	21
5	3	2	5	19	34	22	46	30	17	8	41	21
6	3	2	5	19	34	22	46	30	17	8	41	21
7	3	2	5	19	34	22	46	30	17	8	41	21
8	3	2	5	8	17	19	34	22	46	30	41	21
9	3	2	5	8	17	19	34	22	41	30	46	21
10	3	2	5	8	17	19	34	21	41	22	46	30

gap - - (1 now)

1	3	2	5	8	17	19	34	21	41	22	46	30
2	2	3	5	8	17	19	34	21	41	22	46	30
3	2	3	5	8	17	19	34	21	41	22	46	30
4	2	3	5	8	17	19	34	21	41	22	46	30
5	2	3	5	8	17	19	34	21	41	22	46	30
6	2	3	5	8	17	19	34	21	41	22	46	30
7	2	3	5	8	17	19	21	34	41	22	46	30
8	2	3	5	8	17	19	21	34	41	22	46	30
9	2	3	5	8	17	19	21	34	41	22	46	30
10	2	3	5	8	17	19	21	34	22	41	46	30
In for 3rd loop	2	3	5	8	17	19	21	22	34	41	46	30
In for 3rd loop	2	3	5	8	17	19	21	22	34	41	30	46
In for 3rd loop	2	3	5	8	17	19	21	22	34	30	41	46
In for 3rd loop	2	3	5	8	17	19	21	22	30	34	41	46
11	2	3	5	8	17	19	21	22	30	34	41	46

gap - - (0 now)

GAP = 3

Pass												
1	34	19	3	22	5	2	46	30	17	8	41	21
2	22	19	3	34	5	2	46	30	17	8	41	21
3	22	5	3	34	19	2	46	30	17	8	41	21
4	22	5	2	34	19	3	46	30	34	8	41	21
5	22	5	2	34	19	3	46	30	34	8	41	21
6	22	5	2	34	19	3	46	30	34	8	41	21
7	22	5	2	34	19	3	46	30	34	8	41	21
8	22	5	2	34	19	3	46	30	34	8	41	21
9	22	5	2	34	19	3	8	30	34	46	41	21
in 3rd for()	8	5	2	22	19	3	34	30	17	46	41	21
10	8	5	2	22	19	3	34	30	17	46	41	21
11	8	5	2	22	19	3	34	30	17	46	41	21

gap - - (2 now)

1	2	5	8	22	19	3	34	30	17	46	41	21
2	2	5	8	22	19	3	34	30	17	46	41	21
3	2	5	8	22	19	3	34	30	17	46	41	21
4	2	3	8	5	19	22	34	30	17	46	41	21
5	2	5	8	22	19	3	34	30	17	46	41	21
6	2	5	8	22	19	3	34	30	17	46	41	21
7	2	5	8	22	19	3	34	30	17	46	41	21
8	2	5	8	22	19	3	17	30	34	46	41	21
9	2	5	8	22	19	3	17	30	34	46	41	21
~~in for 3rd loop~~	~~2~~	~~3~~	~~8~~	~~5~~	~~17~~	~~22~~	~~19~~	~~30~~	~~34~~	~~46~~	~~41~~	~~21~~
10	2	3	8	5	17	22	19	30	34	46	41	21
in for 3rd loop	2	3	8	5	17	22	19	30	34	30	41	46
in for 3rd loop	2	3	8	5	17	22	19	22	34	30	41	46
11	2	3	8	5	17	21	19	22	34	30	41	46

gap - - (1 now)

1	2	3	8	5	17	21	19	22	34	30	41	46
2	2	3	8	5	17	21	19	22	34	30	41	46
3	2	3	5	8	17	21	19	22	34	30	41	46
4	2	3	5	8	17	21	19	22	34	30	41	46
5	2	3	5	8	17	21	19	22	34	30	41	46
6	2	3	5	8	17	19	21	22	34	30	41	46
7	2	3	5	8	17	19	21	22	34	30	41	46
8	2	3	5	8	17	19	21	22	34	30	41	46
9	2	3	5	8	17	19	21	22	30	34	41	46
10	2	3	5	8	17	19	21	22	30	34	41	46
11	2	3	5	8	17	19	21	22	30	34	41	46

gap - - (0 now)

Problem 3 [4 points]

Banks often record transactions on an account in the order of the times of the transactions, but many people like to receive their bank statements with checks listed in order by check number. Because people will write checks in order by number and merchants will tend to cash them rather promptly, the problem is one of sorting almost-sorted input (converting time-of-transaction ordering to check-number ordering). Considering insertion sort, mergesort, and quicksort, explain which one would be best for the problem.

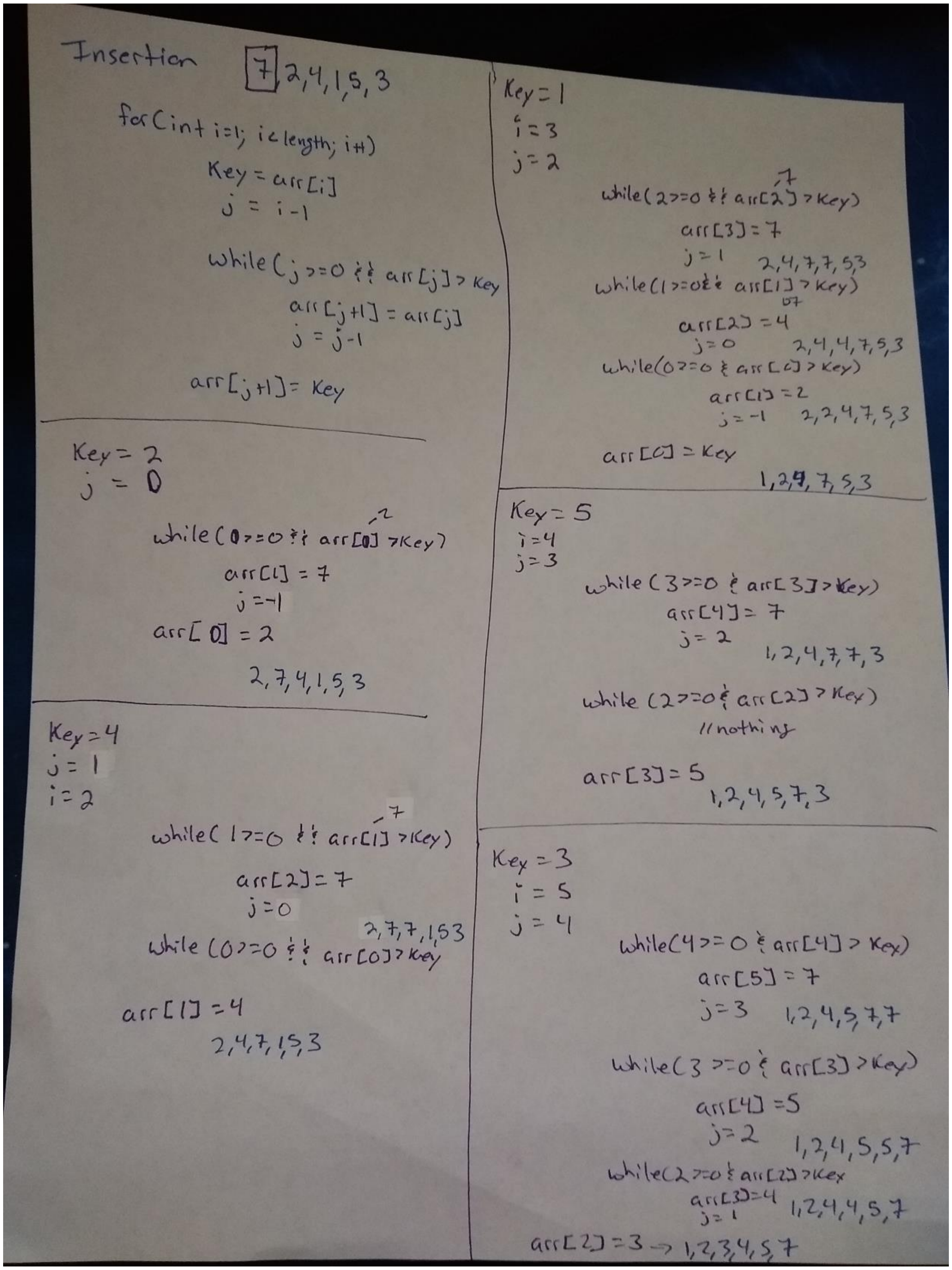
Because MergeSort's sorting algorithm is highly efficient, especially for large data sets (checks for the entire back), it would be more efficient to use this algorithm since it would be able to process the checks faster for an almost "real time" transaction processing. Once sorted, you could quickly select another option to organize the data that doesn't necessarily need almost real-time transaction processing

Reflection [5 points]

In two to three paragraphs of prose (i.e. sentences, not bullet lists) using APA style citations if needed, summarize and interact with the content that was covered in the class "Meet" session (or face-to-face class) this week. In your summary, you should highlight the major topics, theories, practices, and knowledge that were covered. Your summary should also interact with the material through personal observations, reflections, and applications to the field of study. In particular, highlight what surprised, enlightened, or otherwise engaged you. Make sure to include at least one thing that you're still confused about. In other words, you should think and write critically not just about what was presented but also what you have learned through the session. Feel free to ask questions in this as well since it will be returned to you with answers.

Insertion Sort

- Official Definition (Source): Insertion sort is a simple sorting algorithm that builds the final sorted array one item at a time. It is much less efficient on large lists than more advanced algorithms such as quicksort, heapsort, or merge sort.
  - **Worst Complexity:** n^2
  - **Average Complexity:** n^2
  - **Best Complexity:** n
  - **Space Complexity:** 1
- My Notes: Not the best performance sorting algorithm, but it is pretty easy to understand and implement. Separate array into "sorted" and "unsorted" list, if you find a entry larger than the last sorted array section, you shift everything over to the right until the unsorted element is less than the sorted element and add it right after that element.
  - Set element 0 as the "sorted array"
  - Set i to 1 (for int i = 1; i < length – 1; i++)
    - Set value to to array[i]
    - Set "hole" (to move elements over) to i - 1
  - While (hole > 0 && array[hole] > value)
    - array[hole + 1] = array[hole] // move element over one
    - hole = hole – 1
  - array [hole + 1] = key



Insertion $\boxed{7}$,2,4,1,5,3

```
for(int i=1; i<length; 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
```

Key = 2
j = 0

while(0>=0 && arr[$\overset{2}{0}$] >Key)
arr[1] = 7
j = -1
arr[0] = 2

2,7,4,1,5,3

Key = 4
j = 1
i = 2

while(1>=0 && arr[$\overset{7}{1}$] >Key)
arr[2] = 7
j = 0
2,7,7,1,5,3
while (0>=0 && arr[0]>Key

arr[1] = 4
2,4,7,1,5,3

Key = 1
i = 3
j = 2

while(2>=0 && arr[$\overset{7}{2}$] >Key)
arr[3] = 7
j = 1 2,4,7,7,5,3
while(1>=0 && arr[$\overset{4}{1}$] >Key)
arr[2] = 4
j = 0 2,4,4,7,5,3
while(0>=0 & arr[0] >Key)
arr[1] = 2
j = -1 2,2,4,7,5,3

arr[0] = Key
1,2,4,7,5,3

Key = 5
i = 4
j = 3
while (3>=0 & arr[3] >Key)
arr[4] = 7
j = 2
1,2,4,7,7,3

while (2>=0 & arr[2] >Key)
//nothing

arr[3] = 5
1,2,4,5,7,3

Key = 3
i = 5
j = 4

while(4>=0 & arr[4] > Key)
arr[5] = 7
j = 3 1,2,4,5,7,7

while(3 >=0 & arr[3] >Key)
arr[4] = 5
j = 2 1,2,4,5,5,7
while(2>=0 & arr[2] >Key
arr[3] = 4
j = 1 1,2,4,4,5,7
arr[2] = 3 $\rightarrow$ 1,2,3,4,5,7

Merge Sort

- Official Definition (Source): In computer science, merge sort is an efficient, general-purpose, comparison-based sorting algorithm. Most implementations produce a stable sort, which means that the order of equal elements is the same in the input and output. Merge sort is a divide and conquer algorithm.
    - **Worst Complexity:** n(log n)
    - **Average Complexity:** n(log n)
    - **Best Complexity:** n(log n)
    - **Space Complexity:** n
- My Notes: Recursive and pretty efficient sorting algorithm. Merge Sort left half, Merge Sort right half, and then merge left and right in sorted order. Extra space is required since you have to copy all the items into another array, but it's efficient.
    - Set middle to (0 + length) / 2 – this helps up split halves when ordering left half and right half.
    - Call mergeSort on list with 0 to middle
    - Call mergeSort on list with middle + 1 to length -1
    - Call mergeSort on list with 0, middle + 1, and length – 1

Quick Sort

- Official Definition (Source): Quicksort is an O efficient sorting algorithm, serving as a systematic method for placing the elements of a random access file or an array in order.
    - **Worst Complexity:** n^2
    - **Average Complexity:** n(log n)
    - **Best Complexity:** n(log n)
    - **Space Complexity:** n
- My Notes: Efficient if pivot is picked correctly. Pivot is usually, first, last, or median index. The most efficient pivot is usually the median since it's rare that numbers in an array aren't similar in range. Also used recursively because you cycle through different sections of the array depending on pivot.

Shell Sort

- Official Definition (Source): Shellsort, also known as Shell sort or Shell's method, is an in-place comparison sort. It can be seen as either a generalization of sorting by exchange or sorting by insertion. The method starts by sorting pairs of elements far apart from each other, then progressively reducing the gap between elements to be compared.
    - **Worst Complexity:** Depends on gap sequence
    - **Average Complexity:** $n(\log n)^2$ or $n^{3/2}$
    - **Best Complexity:** n
    - **Space Complexity:** n
- My Notes: Depending on the gap, it can be a very efficient algorithm. The usual gap is length / 2. After the elements are compared by the gap, the rest is basic insertion