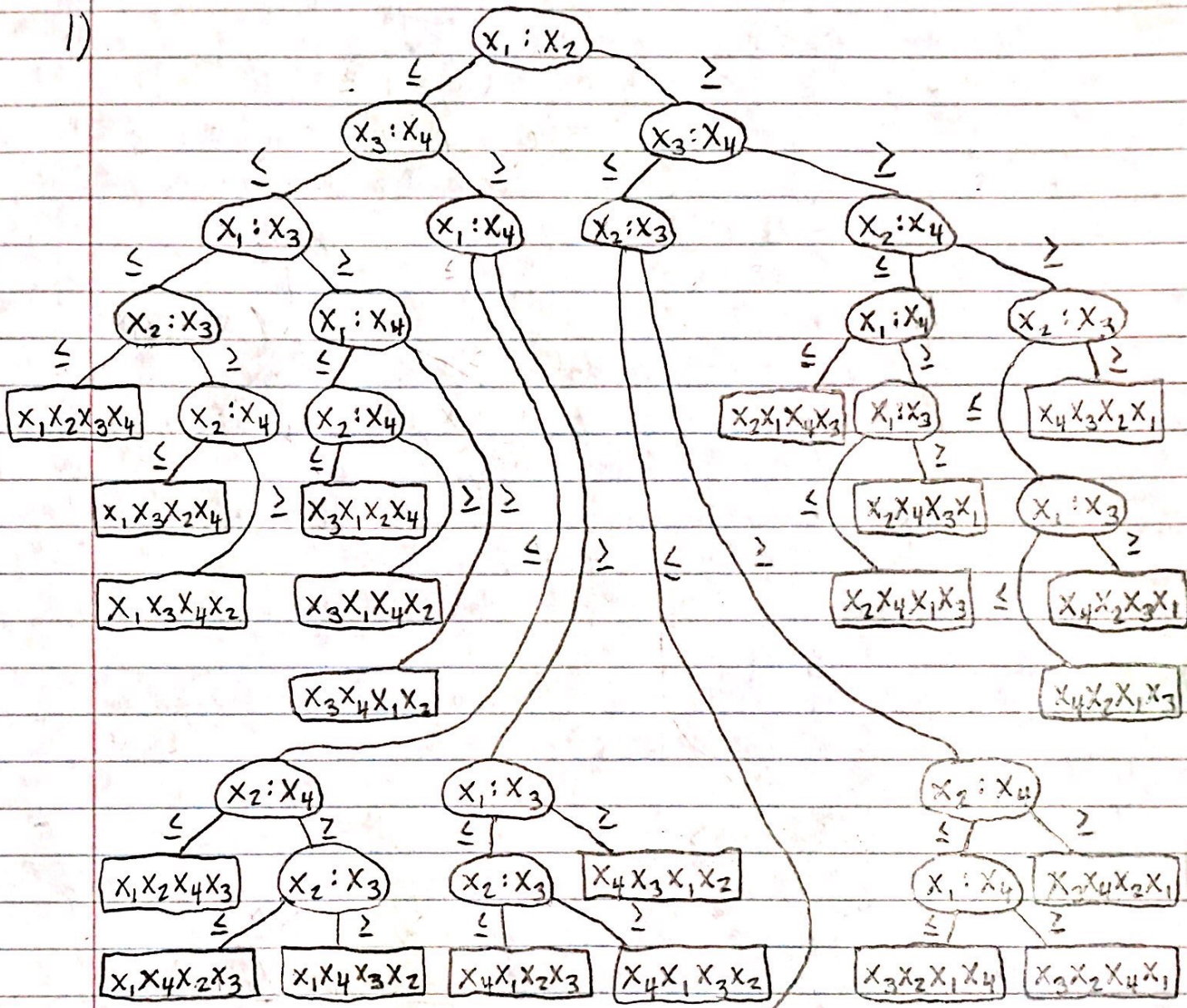
CS 4040 - HW 3

Ethan
Dowalter

1)



This tree represents a decision tree for the Merge Sort algorithm, The height = 5 = $\lceil \log_2 4! \rceil$.

Average comparisons = $4(\frac{8}{24}) + 5(\frac{16}{24})$

$= \frac{112}{24} = \frac{14}{3} \approx 4.67$

The best case is 4 comparisons

The worst case is 5 comparisons

2) Function mySort(array[n]){
    for (i = 0; i < n; i++){
        array[i] = convertToBaseN(array[i], n)
            //Convert each element from base 10 to base n}
    RadixSort(array)
}
This function will sort n integers in the range 1 to n^3 in O(n) time because convertToBaseN takes constant time and is run n times. Then, RadixSort is guaranteed to run in O(n) time because convertToBaseN makes it so every element has at most 4 digits (because n^3), so it would take about O(4n) time. Therefore, O(n)+O(4n) = O(n) ✓

3) a) Quicksort - Unstable because it exchanges nonadjacent elements
Ex: $(4_1, 2, 3, 4_2, 1) \rightarrow (1, 2, 3, 4_2, 4_1)$
Pivot on 3

b) Bubble sort - Stable because it only swaps elements if they are >, so equal elements are never swapped

c) Insertion sort - Stable because it places elements from left to right and only if they are >

d) Counting sort - Stable because it retains the ordering of equal elements, but in a right-to-left fasion

e) Heap sort - Unstable, equal elements will always be reversed
Ex: $(3_1, 3_2, 2, 1) \rightarrow (1, 2, 3_2, 3_1)$

f) Bucket sort - Stable as long as the buckets are sorted using a stable algorithm

g) Shell sort - Unstable because it doesn't examine elements between intervals
Ex: $(3_1, 3_2, 1, 4) \rightarrow (1, 3_2, 3_1, 4)$

Ethan Dowalter

CS 4040 - HW 3 (cont.)

	Start	Step 1 ↓	Step 2 ↓	Step 3 ↓	Step 4 ↓
4)	THIG	ACTA	SCAG	BABA	ACNE
	PLOD	BABA	AWAY	BABE	ACRE
	BACK	PLOD	BABA	BABU	ACTA
	EVIL	EDGE	BABE	BABY	ACTS
	SCAG	DOGE	BABU	BACK	AWAY
	EDGE	BABE	BABY	RANG	BABA
	YEGG	ACRE	BACK	SCAG	BABE
	PLOT	ACNE	EDGE	ACNE	BABU
	DOGE	DOPE	DOGE	ACRE	BABY
	ACTS	THIG	YEGG	ACTA	BACK
	PLOW	SCAG	THIG	ACTS	DOGE
	AWAY	YEGG	EVIL	EDGE	DOPE
	RANG	RANG	FILL	YEGG	EDGE
	BABE	WING	ACNE	THIG	EVIL
	ACRE	PLUG	RANG	FILL	FILL
	WING	BACK	WING	WING	PLOD
	BABY	EVIL	PLOD	PLOD	PLOP
	PLUG	FILL	PLOP	PLOP	PLOT
	ACTA	PLOP	PLOT	PLOT	PLOW
	BABU	ACTS	PLOW	PLOW	PLUG
	PLOP	PLOT	DOPE	PLUG	RANG
	FILL	BABU	ACRE	DOGE	SCAG
	ACNE	PLOW	ACTA	DOPE	THIG
	DOPE	AWAY	ACTS	EVIL	WING
	BABA	BABY	PLUG	AWAY	YEGG

5) You can simply use Radix sort to accomplish this You need to slightly modify the algorithm however, because it's possible that not all elements have the same number of digits. In the case where we run out of digits on a given element, you just treat that as a -1 and send it to the "first bucket" or front of the list, and remove it from further sorting steps. So, if there are n total digits in the array, K is the maximum number of digits of an element, and m is the number of elements, this modified Radix sort would run in O(n) time due to the handling of numbers with fewer than K digits.

```
Function modifiedRadixSort(array[]){
    K = getMaxDigits(array)
    m = array.length()
    for(i=0; i<K; i++){
        count[11] = {0}    // All 0's
        for(j=0; j<m; j++){
            count[(Digit at array[j] in pass i)+1]++
            // If no digit, use count[0] }
        for(j=1; j<11; j++){
            count[j] = count[j] + count[j-1] }
        result[m] = {0}
        for(j=m-1; j≥0; j--){
            result[count[Digit at array[j]]] = array[j]}
        for(j=0; j<m; j++){
            array[j] = result[j] } //Put results in array
    }
}
```

CS 4040 - HW 3 (cont.

Ethan Dowalter

6) a) The second iteration of the while loop effectively mimics the second recursive call in QuickSort because of the line p=q+1. This means that Quicksort and Tail-Recursive-Quicksort sort the array in the same manner, which implies Tail-Recursive-Quicksort is correct because we know Quicksort is correct

b) If the array is already sorted then the right sub-array in Partition will always have size 0, which would force there to be n calls to Tail-Recursive-QuicKsort, implying $\Theta(n)$ performance

c)
```
Tail-Recursive-QuicKsort-Modified(A, p, r)
    while p < r
        q = Partition(A, p, r)
        if q < ⌊(p+r)/2⌋
            Tail-Recursive-QuicKsort-Modified(A, p, q-1)
            p = q + 1
        else
            Tail-Recursive-Quicksort-Modified(A, q+1, r)
            r = q - 1
```

7)
```
findTopKPercent(earnings[n], K){   //(0 ≤ K ≤ 100)
    cutoffIndex = ⌊n * ((100-K)/100)⌋ // starting index of top K%
    cutoffValue = Select(earnings, cutoffIndex, 0, n)
    topKPercent[n - cutoffIndex] = {0} //Initialize result array
    j = 0
    for (i = 0; i < n ; i++)
        if earnings[i] ≥ cutoffValue
            topKPercent[j] = earnings[i]
            j++
    return topKPercent
}
```

This algorithm is optimal because it runs in $O(n)$ time. This is because Select, which finds the cutoffValue for the top K% of earners (i.e. the minimum value which it takes to be part of the top K%), runs in $O(n)$ time, and the for loop which loads the output array, topKPercent[], with all the values that are $\geq$ cutoffValue also runs in $O(n)$ time. Therefore, the whole function runs in $O(n)$ time.

8) a) Select with groups of 3:

After finding s (median of medians), there are $2(\lceil \frac{1}{2} \lceil \frac{n}{3} \rceil \rceil - 2) > \frac{1}{3}n - 4$ elements $> s$.

Similarly, there are at least $\frac{1}{3}n - 4$ elements $< s$.

So, $n - (\frac{1}{3}n - 4) = \frac{2}{3}n + 4$, and

$T(n) \leq T(\frac{2}{3}n+4) + T(\lceil \frac{n}{3} \rceil) + \theta(n)$

Prove $T(n) = O(n)$

I.H.

Assume $T(n) \leq T(\frac{2}{3}n+4) + T(\lceil \frac{n}{3} \rceil) + \theta(n) = O(n) \leq cn - d \ \forall n > n_0$

I.S.

Prove $T(m) \leq T(\frac{2}{3}m+4) + T(\lceil \frac{m}{3} \rceil) + \theta(m) = O(m) \leq am - b \ \forall m > n$

$T(m) \leq a(\frac{2}{3}m+4) - b + a\lceil \frac{m}{3} \rceil - b + xm \leq am - b$

$T(m) \leq \frac{2}{3}am + 4a - b + a(\frac{m}{3}+1) - b + xm \leq am - b$

$T(m) \leq \frac{2}{3}am + 4a - b + \frac{1}{3}am + a - b + xm \leq am - b$

$T(m) \leq am + 5a - 2b + xm \leq am - b$

This inequality is true provided the constants a, b, and x have this relationship - $b \geq 5a + xm$

Therefore, $T(m) = O(m)$ and Select with groups of 3 runs in linear time.

CS 4040 - HW 3 (cont.)

Ethan Dowalter

8) b) Select with groups of 7:

After finding s (median of medians), there are $4(\lceil \frac{1}{2}\lceil \frac{n}{7}\rceil\rceil - 2) > \frac{2}{7}n - 8$ elements $> s$.

Similarly, there are at least $\frac{2}{7}n - 8$ elements $< s$.

So, $n - (\frac{2}{7}n - 8) = \frac{5}{7}n + 8$, and

$T(n) \leq T(\frac{5}{7}n + 8) + T(\lceil \frac{n}{7}\rceil) + \Theta(n)$

Prove $T(n) = O(n)$

I.H.

Assume $T(n) \leq T(\frac{5}{7}n+8) + T(\lceil \frac{n}{7}\rceil) + \Theta(n) = O(n) \leq cn - d \quad \forall n > n_0$

I.S.

Prove $T(m) \leq T(\frac{5}{7}m+8) + T(\lceil \frac{m}{7}\rceil) + \Theta(m) = O(m) \leq am - b \quad \forall m > n$

$T(m) \leq a(\frac{5}{7}m+8) - b + a\lceil \frac{m}{7}\rceil - b + xm \leq am - b$

$T(m) \leq \frac{5}{7}am + 8a - b + a(\frac{m}{7}+1) - b + xm \leq am - b$

$T(m) \leq \frac{5}{7}am + 8a - b + \frac{1}{7}am + a - b + xm \leq am - b$

$T(m) \leq \frac{6}{7}am + 9a - 2b + xm \leq am - b$

$T(m) \leq am - \frac{1}{7}am + 9a - 2b + xm \leq am - b$

This inequality is true provided the constants a, b, and x have this relationship - $b \geq 9a + xm - \frac{1}{7}am$

Therefore, $T(m) = O(m)$ and Select with groups of 7 also runs in linear time.