Michael Patson
HW 2 CS 325

Problem 1

a) $T(n)=3T(n-1)+1$

t(n) = 3T(n-1) +1

3^2 t(n-2)+3+1

3^3 t(n-3)+3^2+3+1

This forms a series that can be represented by **Θ(3^n)**

b) $T(n) = 2T\left({}^{n}/_{4}\right) + nlgn$;

Using the master method

a=2, b=4, f(n) =nlog(n)

$\log_b(A)= \log_4 2 =1/2$

n^(1/2)

f(n) = nlogn > n^(1/2)

there for t(n)= O(nlogn)

Michael Patson
HW 2 CS 325

Problem 2

a) Must assume array is sorted. Instead of dividing in half like binary search, we will divide in thirds. The search will check against both “midpoints” or third way markers. It will only check 1/3 of the list each time.

Below is pseudo-code

n is elements in array, r is n-1 to give right bound. l is equal to 0, first position in array, x is the search value

```
ternSearch(arr, l, r, x)

if (r>=l)

            third1 = l + (r - l)/3;

            third2 = (l + (r - l)/3) * 2;

if (arr[third1] = x) then return third1; //we had a direct hit between 1/3 and 2/3

 if (arr[third2] = x) then return third2; //had a hit a direct divided between 2/3 and 3/3

//now check each third

     if (arr[third1] >x)

            return return ternSearch(arr, l, third1 -1, x); // recursively checks the first third

     if (arr[third2] < x)

            return ternSearch(arr, third2+1, r, x);

else

//set new left and right bounds with middle search

return ternSearch(arr, third+1, third2-1, x);

  }

  return -1 ;//breaks if no element is found
```

b) The runtime would be

Michael Patson
HW 2 CS 325

$T(n) = T(n/3)+4$

using the master theory

a=1, b=3/1, d=0

so, $O(Log_3N)$ for time complexity

c) we know binary search is $O(log_2n)$

In full form, for comparison we see

binary = $2log_2n + O(1)$

tertiary = $4log_3n + O(1)$

reduce terms out

bin = $log_2n$; tert =2 $log_3n$ we can see that binary is more efficient. To double check, one can evaluate with n= 100 and 10000, binary is ~6.6 and ~13.6 with tertiary at ~8.3 and ~16.7.

Therefore, we can see **binary** is more efficient of a search. Another way to think about it is like this, how can I get rid of most of my data quickly? If you have a list 1-30 and you are looking for 15, if you do binary you can throw out 1-15 in one step. It takes more steps to throw out 1-10, 11-20.

For example

1-30 n=2 would be 1-15 and 16-30

1-30 n=3 would be 1-10, 11-20 and 21-30

1-30 n=5 would be 1-6, 7-12, 13-18, 19-24, 25-30

1-30 n=30 would be 1, 2, 3, 4... etc becoming just a linear search

Michael Patson
HW 2 CS 325

Problem 3

A recursive pseudo code for max and min. We will input an unsorted array. This is compared to the naïve method, the divide and conquer approach split the array in half like the binary search.

The idea is to divide the problem in half and solve each half. Then to combine the results.

```
MinMax(arr, min, max)
If right=left                    //if only one element
            min=max=left
else if right  - left =1
if arr[left]<=arr[right]        //if only two elements
            min=arr[right]
            max = arr[left]
else
            min=arr[right]
            max =arr[left]
else if right-left > 1     //many n elements, recursive
            MinMax (arr[left]…[(left+right)/2], min, max)  left half of array min and max
            MinMax (arr [(left+right)/2]+1….right], min2, max2) right half of array min and max
//combine results
if min2<min
            min=min2
if max2 >max
            max=max2
return min and max
```

b. When there is one term, no comparison T(1)=0. When two, T(2) =1, see commented code above.

If there are more than two elements, since each half gets half the work,
Recurrence is T(n) = 2T(n/2) + 2

c. $T(n) = 2T(n/2) + 2$
$= 2(2T(n/4) + 2)+2$
$=2^2T(n/4)+2*2+2$
$=2^3T(n/2\text{^}3)+2\text{^}3+3\text{^}2+2$

=2^(d-1) T(2) +2^(d-1)+2^(d-2)
=2^d+2^(d-1)-2
=2^d+(1/2)2^d-2
=n+1/2(n)-2
=(3/2)n-2

Michael Patson
HW 2 CS 325

Problem 4

| n | Comparison | Calculation |
|---|---|---|
| 1 | 0 | |
| 2 | 1 | |
| 3 | 3 | 3*1+0=3 |
| 4 | 9 | 4*2+1 |
| 5 | 17 | 5*3+2 |

The comparisons are n*(n-2)*(n-3)

So recurrence is T(n) 3T(3n/2)= Θ(1)

Using the Master's Theorem

T(n)=aT(n/b)+f(n)

a=3, b=2/3 (fn)=1

n^$\log_{(2/3)}3$

so $O(n^{2.71})$

Michael Patson
HW 2 CS 325

Problem 5

b)

| Items in Array | Run Time 1 | Run Time 2 | Average |
|---|---|---|---|
| 1,000 | 0.96 | 1.03 | 0.995 |
| 2,000 | 5.5 | 5.94 | 5.720 |
| 3,000 | 16.02 | 15.82 | 15.920 |
| 4,000 | 46.63 | 46.85 | 46.740 |
| 5,000 | 48.1 | 47.6 | 47.850 |
| 6,000 | 139.700 | 139.500 | 139.600 |
| 10,000 | 417.200 | 417.800 | 417.500 |

Figure 1. Above are the runtimes for the merge sort

c)

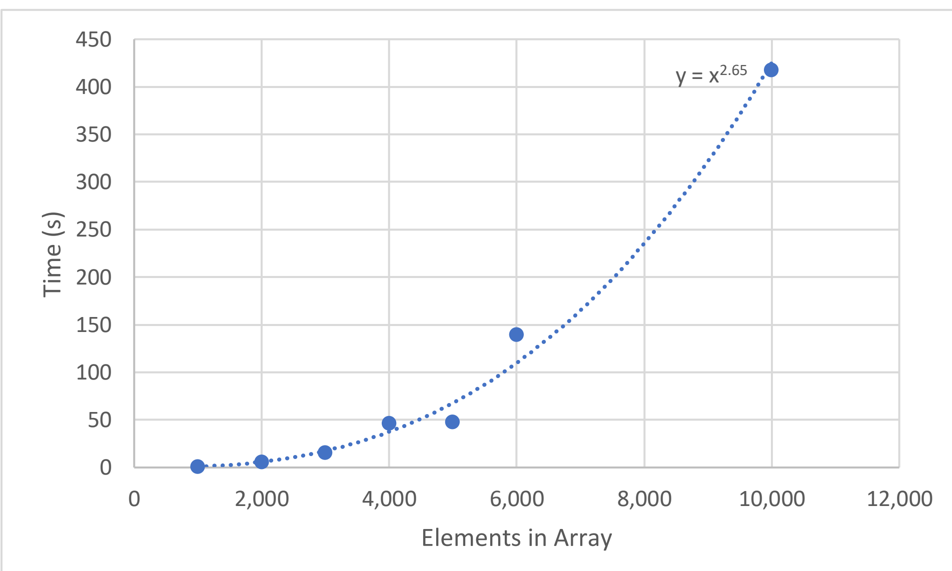


Figure 2 shows Stooge Sort experiment run times with a fit line $y = x^{2.65}$.

Michael Patson
HW 2 CS 325

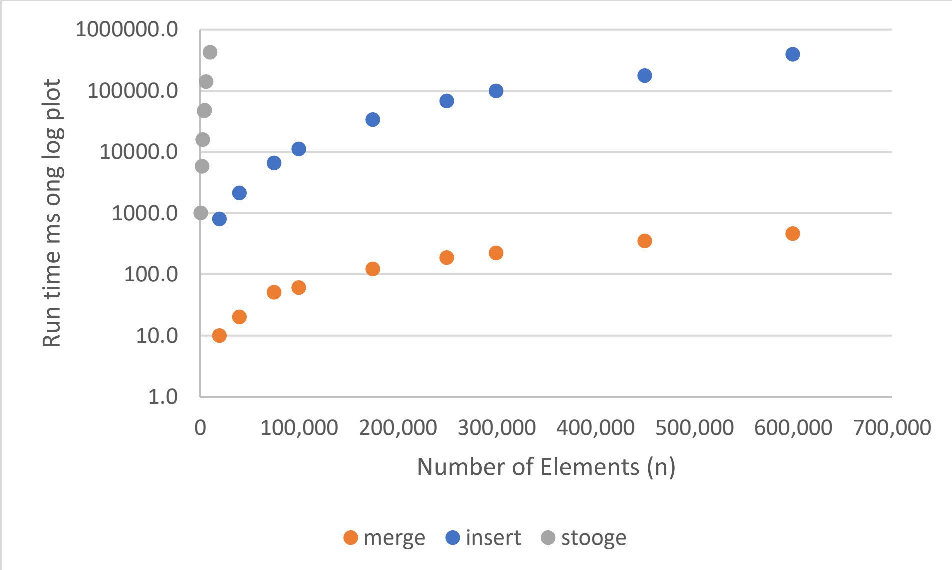


Figure 3 Shows a merge, insert and stooge sorts on a log log plot. Stooge times were so large that large numbers of elements in the array could not be recorded.

The data above shows that the stooge sort runs inefficiently compared to the other types of sorts. Additionally, the line that fits the experimental stooge data is $y=x^{2.65}$. The theoretical value should be $y=x^{2.7}$. If a large data set were able to be run, I would expect that the data would conform more closely to the theoretical data.

Michael Patson
HW 2 CS 325

Code:

```
/*******

Name: Michael Patson

Homework Assignment 2



Stoogesort pseudocode

StoogeSort(A[0 ... n - 1])

if n = 2 and A[0] > A[1]

swap A[0] and A[1] else if n > 2

m = ceiling(2n/3) StoogeSort(A[0 ... m - 1])

StoogeSort(A[n - m ... n - 1])

Stoogesort(A[0 ... m - 1])

**********/




#include<stdlib.h>

#include<stdio.h>

#include<iostream>

#include<fstream>

#include<cstring>

#include<sstream>

#include<math.h>

#include<string.h>

using namespace std;
```

Michael Patson
HW 2 CS 325

```
void STOOGESORT(int arr[], int left, int h)
{
  int n = h-left+1;

  if ((n==2) &&(arr[left]>arr[h]) )
    swap(arr[left],arr[h]);
  else if (n > 2)
  {
    //split into thirds(calcs third way)
    int third= floor(n/3);

    //recurive yosrt first 2/3
    STOOGESORT(arr, left, (h - third));
    //recurive yosrt last 2/3
    STOOGESORT(arr, (left+third), h);
      //sort first 2/3 again
     STOOGESORT(arr, left, (h - third));
  }

}

void printArray(int arr[], int size)
{
        int i;
        for (i=0; i < size; i++)
        printf("%d ", arr[i]);
        printf("\n");
}

int main()
{

  clock_t start_t, end_t, total_t;

  //make array, enter N here
  int n;
  cout<< "PLease enter number of items to generate: ";
  cin>> n;
  int i;
  int count;
  int arr[n];


  for (int x =0; x<n; x++)
  {
    i= rand() % 1000;
   arr[x]= i;
    count++;
  }
        start_t = clock();
```

Michael Patson
HW 2 CS 325

```
            STOOGESORT(arr,0,(n-1));
            printArray(arr,n);

  end_t = clock();
   printf("End of the run, end_t = %ld\n", end_t);

   float diff ((float) end_t-(float)start_t);
   float seconds = diff/ CLOCKS_PER_SEC;

  printf("Total time taken by CPU: %f\n", seconds  );

   cout<<"Number of elements in Array is : ";
   cout<< n;
   printf("\n");
   cout<<"Confirmed of elements in Array is : ";
   cout<< count;
   printf("\n");

   return 0;
}
```