Emiliano Manalo

CSC-17C Midterm

1. Linear Vs Binary Search Analysis

int binSrch(int a[],int n, int val){

//Initialize end points which are indexes

int lowEnd=0;

int highEnd=n-1;

//Loop until value found not indexes

do{



int middle=(highEnd+lowEnd)/2;



if(val==a[middle])return middle;



else if(val>a[middle])lowEnd=middle+1;



else highEnd=middle-1;



}while(lowEnd<=highEnd);



//Not found

return -1;

}

int linSrch(int a[],int n, int val){

for(int indx=0;indx<n;indx++){



if(val==a[indx])return indx;



}

return -1;

}

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *Operational* |  |  |  |  |  | Timings | Binary | Linear |
| **N** | O(N) | Log(2, N) | Binary | Linear |  | N | ms | ms |
| 10 | 10 | 3.32192809 | 6 | 10 |  | 1000 |  | 0.00157 |
| 25 | 25 | 4.64385619 | 7 | 18 |  | 2000 | 0.000152 | 0.001552 |
| 50 | 50 | 5.64385619 | 10 | 43 |  | 5000 |  | 0.002521 |
| 100 | 100 | 6.64385619 | 11 | 83 |  | 7500 |  | 0.002886 |
| 200 | 200 | 7.64385619 | 9 | 118 |  | 10000 |  | 0.004557 |

1. Bubble Sort Vs Selection Sort Analysis

void bublSrt(int a[],int n){

//Keep looping and comparing until no swaps are left

bool swap;

do{

swap=false;

//Check the list and Swap when necessary

for(int i=0;i<n-1;i++){

if(a[i]>a[i+1]){

int temp=a[i];



a[i]=a[i+1];

a[i+1]=temp;

swap=true;

}

}



}while(swap);



}



|  |  |  |  |
| --- | --- | --- | --- |
| Operational |  |  |  |
| N | N^2 | Bubble Sort | Selection Sort |
| 100 | 10000 | 18551 | 5546 |
| 200 | 40000 | 74265 | 21186 |
| 300 | 90000 | 172807 | 46842 |
| 400 | 160000 | 301731 | 82514 |
| 500 | 250000 | 483074 | 128166 |

|  |  |  |
| --- | --- | --- |
| Timing |  |  |
| N | Bubble Sort | Selection Sort |
| 100 |  |  |
| 200 |  | 4300 |
| 400 | 58200 | 86900 |
| 800 | 1076900 | 1529500 |
| 1600 | 5919300 | 2899700 |

void selSort(int a[],int n){

//Loop and declare variables

int indx,min;

for(int pos=0;pos<n-1;pos++){

//Find the smallest in the list, swap after finding

min=a[pos];indx=pos;

for(int i=pos+1;i<n;i++){

if(a[i]<min){



min=a[i];



indx=i;



}



}



//Perform the swap



a[indx]=a[pos];

a[pos]=min;



}



}



1. Push Method
2. void SimpleVector<T>::push(T &val){
3. //Declare new array pointer
4. T \*naptr;
5. // Allocate memory for the new array.
6. try{
7. naptr = new T [arraySize+1];
8. }catch (bad\_alloc){
9. memError();
10. }
11. //Fill the Array



1. for (int count = 0; count < arraySize; count++){



1. naptr[count]=aptr[count];
2. }
4. //Add new value and increment the array size
5. naptr[arraySize++]=val;
7. //Delete the old array and point to new array
8. delete []aptr;
9. aptr=0;
10. aptr=naptr;
11. }



Optimized Simple Vector

void SimpleVector<T>::push(T &val){

    if(arraySize==maxSize){

        maxSize\*=2;

        //Declare new array pointer

        T \*naptr;

        // Allocate memory for the new array.

        try{

          naptr = new T [maxSize];

        }catch (bad\_alloc){

          memError();

        }

        //Fill the Array

        for (int count = 0; count < arraySize; count++){



            naptr[count]=aptr[count];

        }



        //Add new value and increment the array size



        naptr[arraySize++]=val;

        //Delete the old array and point to new array

        delete []aptr;

        aptr=0;

        aptr=naptr;

    }else{

        aptr[arraySize++]=val;

    }

template <class T>

void SimpleVector<T>::push(const T &val){

    //Add a value into the Linked List

    aptr->addLst(val);



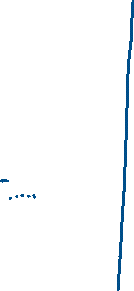
}

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Operational |  |  |  |  |
| N | N | Simple Vec | OP Simple Vec | Linked List |
| 100 | 100 | 100 |  | 1 |
| 200 | 200 | 200 |  | 1 |
| 300 | 300 | 300 |  | 1 |
| 400 | 400 | 400 |  | 1 |
| 500 | 500 | 500 |  | 1 |

|  |  |  |  |
| --- | --- | --- | --- |
| Timings |  |  |  |
| N | Simple Vec | OP Simple Vec | Linked List |
| 100 |  |  |  |
| 200 | 28.421 |  |  |
| 300 | 60.195 |  |  |
| 400 | 108.9 |  |  |
| 500 | 183.27 |  |  |

1. Big(O) Graph

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Big O()** | ***Second*** | ***Minute*** | ***Hour*** | ***Day*** | ***Month*** | ***Year*** | ***Decade*** | ***Century*** |
| *N^1/3* | 2.10E+06 | 1.26E+08 | 7.55E+09 | 1.81E+11 | 5.44E+12 | 6.52E+13 | 6.52E+14 | 6.52E+15 |
| *N^1/2* | 1.64E+04 | 9.83E+05 | 5.90E+07 | 1.42E+09 | 4.25E+10 | 5.10E+11 | 5.10E+12 | 5.10E+13 |
| *N* | 1.28E+02 | 7.68E+03 | 4.61E+05 | 1.11E+07 | 3.32E+08 | 3.98E+09 | 3.98E+10 | 3.98E+11 |
| *Nlog(N)* | 6.07E+01 | 3.64E+03 | 2.19E+05 | 5.25E+06 | 1.57E+08 | 1.89E+09 | 1.89E+10 | 1.89E+11 |
| *N^2* | 1.13E+01 | 6.79E+02 | 4.07E+04 | 9.78E+05 | 2.93E+07 | 3.52E+08 | 3.52E+09 | 3.52E+10 |
| *N^2log(N)* | 7.79E+00 | 4.68E+02 | 2.81E+04 | 6.73E+05 | 2.02E+07 | 2.42E+08 | 2.42E+09 | 2.42E+10 |
| *2^N* | 7.00E+00 | 4.20E+02 | 2.52E+04 | 6.05E+05 | 1.81E+07 | 2.18E+08 | 2.18E+09 | 2.18E+10 |
| *N!* | 5.00E+00 | 7.00E+00 | 9.00E+00 | 1.00E+01 | 1.10E+01 | 1.20E+01 | 1.30E+01 | 1.40E+01 |



1. Non-Recursive vs Recursive Fibonacci Big(O)

int fibLoop(int n){

//Base Case

if(n<=0)return 0;

if(n==1)return 1;



int fim1=1,fim2=0,fi=fim1+fim2;

for(int i=2;i<n;i++){

fim2=fim1;



fim1=fi;



fi=fim1+fim2;

}

return fi;

}



int fibRec(int n){

//Base Case

if(n<=0)return 0;



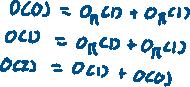
if(n==1)return 1;

//Recursive Representation

return fibRec(n-1)+fibRec(n-2);



}



7.



8.

