**Devansh Gupta**

**01609278**

**CIS 455 – Homework #1**

**Problem 1-1.**Write an algorithm that, given a list of *n* numbers, returns the largest and smallest numbers in the list.

Estimate the running time of the algorithm.

Can you design an algorithm that performs only 3*n*/2 comparisons to find the smallest and largest numbers in the

list?

Solution:

Let S=(x1,x2,....,xn) be the list of no's to sort and consider n to even. Now S contains even no's of elements. We should now divide the list of numbers into two lists i.e. small and large. Now we will search for smallest number from i=0 to n/2 and place the smallest element into small and the largest number in large. Then we will do a linear search in list of small to find the smallest no's and through list of large to find the largest number.

This algorithm will take n/2 comparisons to divide the list of numbers into small list and large list. Each will take less then n/2 comparisons to find the smallest and largest number. So the total comparisons will be n which is less then 3n/2.

Example. Let L = (6,2, 5, 3, 10, 5, 3, 3, 4, 7). We make 5 pair wise comparisons to split the list into small = (2, 3, 5, 3, 4) and large = (6,5, 10, 3, 7). To find the smallest element of small we must compare the first element to each other, so that is 4 comparisons. To find the largest element of large we must compare the first element to each other, so that is also 4 comparisons. Therefore the total number of comparisons is 5+4+4 = 13, which is 3(10)/2-2 as expected.

Code:

//Write an algorithm that, given a list of n numbers, returns the largest and smallest numbers in the list.

#include<stdio.h>

#include<stdlib.h>

int main()

{

int a[10],small[10],large[10];

int i=0,j=0,k=0,l=0,n=0,mid=0,first=0,temp=0;

int sml=0,lrg=0;

for(i=0;i<10;i++)

a[i]=small[i]=large[i]=0;

printf("Please enter the nos of elements in the list:");

scanf("%d",&n);

printf("\nPlease enter numbers in list:");

for(i=0;i<n;i++)

scanf("%d",&a[i]);

printf("\nNumbers in list are ");

for(i=0;i<n;i++)

printf("%d\t",a[i]);

if(n%2==1)

mid=n/2;

first=a[0];

for(i=0;i<n;i++)

if(i==mid && mid!=0 )

printf("\nHey there is a mid and is %d",a[mid]);

else

{

if(a[i]<a[i+1] && i!=i-1)

{

small[k]=a[i];

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

i++,k++,l++;

}

else

if(i!=1)

{

large[l]=a[i];

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

i++,l++,k++;

}

}

printf("\nDivided small list:");

printf("\n");

for(i=0;i<k;i++)

printf("%d\t",small[i]);

printf("\nDivided large list:");

printf("\n");

for(i=0;i<l;i++)

printf("%d\t",large[i]);

printf("\n");

for(i=0;i<k;i++)

for(j=i+1;j<k;j++)

if(small[i]>small[i+1])

{

temp=small[i];

small[i]=small[j];

small[j]=temp;

}

for(i=0;i<l;i++)

for(j=i+1;j<l;j++)

if(large[i]>large[i+1])

{

temp=large[i];

large[i]=large[j];

large[j]=temp;

}

printf("Sorted Smaller list:");

for(i=0;i<k;i++)

printf("%d\t",small[i]);

printf("\n");

printf("Sorted larger list:");

for(i=0;i<l;i++)

printf("%d\t",large[i]);

if(n%2==1)

{

if(small[0]>a[mid])

sml=a[mid];

else

sml=small[0];

if(large[l-1]<a[mid])

lrg=a[mid];

else

lrg=large[l-1];

}

else

{

sml=small[0];

lrg=large[l-1];

}

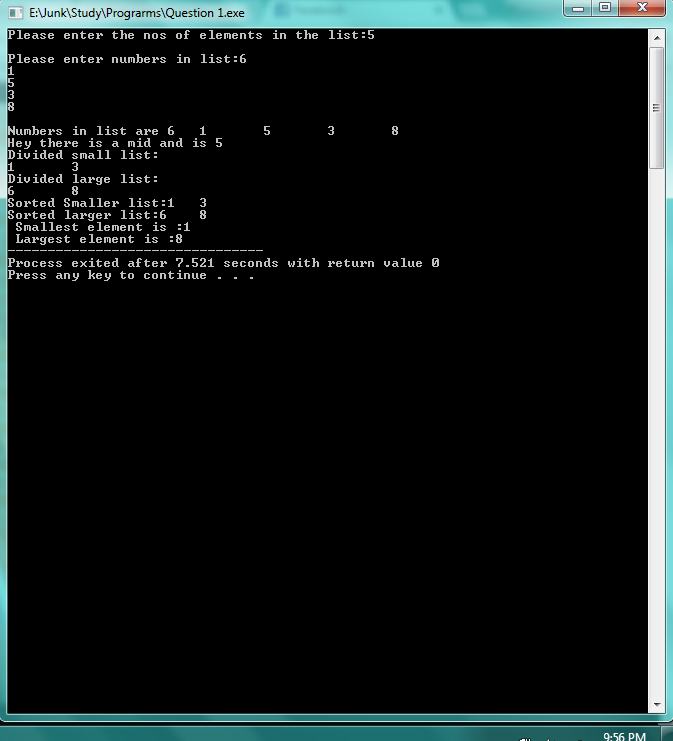
printf("\n Smallest element is :%d",sml);

printf("\n Largest element is :%d",lrg);

return 0;

}

Output:



Pseudo Code:

###Part 1

For dividing and inserting small and big elements of a pair in small list and large list

n<-No's of elements in list

for(i=0 to n)

if i==mid and mid !=0

Skip Count

else

if a[i]<a[i+1] and i!=i-1

{

small[k] <- a[i];

large[l] <- a[i+1];

i++,k++,l++;

}

else

if i!=1

{

large[l] <- a[i];

small[k] <- a[i+1];

i++,l++,k++;

}

Example: Consider List to be S={8,2,5,6,3}

step 1: If mid skip count

step 2:else if compare both elements of a pair to find smaller element and larger element

8 is compared with 2 Since it is the first pair

step 3:Insert large element in large list and small element in small list

8 is large so it will be inserted in large list and 2 in small list

In same way it is done for all other pairs except the mid element

If there is a mid then we skip the element and move to next pair

So the small list={2,3}

large list={6,8}

#####Part 2

Performing sort on small and large list to find smallest from list of small and largest from list of large

k<- No's of elements in small list

l <- No's of elements in large list

for i=0 to k

for j=i+1 to k

if small[i] > small[i+1]

{

temp <- small[i];

small[i] <- small[j];

small[j] <- temp;

}

for i=0 to l

for j=i+1 to l

if large[i]>large[i+1]

{

temp <- large[i];

large[i] <- large[j];

large[j] <- temp;

}

Example: so we get small list and large list from step 1

i.e. small list={2,3} large list={8,6}

step 1:I performs sort on small list to find smallest element and arranges it

small list is sorted so there will be no change

Large list will change to {6,8}

step 2:No if there is a mid then we compare to smallest and largest elements of small list and large list respectively

Here mid = 5 which is neither small then 2 nor greater than 8

so min=2 and max=8

**Problem 1-2.** Write two algorithms that iterate over every index from (0, 0, … , 0) to (*n*1, *n*2, … , *n*d).Make one algorithm recursive, and the other iterative.

Solution:

**The iterative algorithm for this is below :**

**Below code can take any nos of elements in a set and can show maximum nos of known possible sets. It can map all elemets in a list no matter how big or complex.**

#include<stdio.h>

#include<stdlib.h>

int main()

{

int d,g[10],temp[20];

int tem,n,j,i;

printf("Please enter value for d\*\*\*\* i.e nos of lists required:");

scanf("%d",&d);

printf("\nPlease enter nosin set from (0,0,...) to (n1,n2,....):");

for(int i=0;i<10;i++)

{

g[i]=0;

}

for(int i=0;i<d;i++)

{

scanf("%d",&g[i]);

}

for(int i=0;i<20;i++)

{

temp[i]=0;

}

int js=0;

tem=g[0];

int k=0,e=0;

for(i=0;i<tem;i++)

{

for(j=js;j<=i;j++)

{

temp[e]=j+1;

e++;

}

if(i==tem-1)

{

k=k+1;

tem=g[k];

js=0;

j=js;

i=-1;

}

else

js=j;

}

printf("\nElements in temp :");

j=0;

for(i=0;i<d;i++)

{

tem=j+g[i];

j=tem;

}

for(i=0;i<tem;i++)

printf("%d\n",temp[i]);

i=0;

e=0;

int poss=1;

tem=1;

for(int n=0;n<d;n++)

{

poss=tem\*g[n];

tem=poss;

}

printf("Nos of possible sets : %d",tem);

int x;

x=g[0];

int gre=0,sml=0;

for(i=1;i<d;i++)

{

if(x<g[i])

{

gre=g[i];

x=gre;

}

else

gre=x;

}

x=g[0];

printf("\nGreatest element is %d",gre);

for(i=1;i<d;i++)

{

if(x>g[i])

{

sml=g[i];

x=sml;

}

else

sml=x;

}

printf("\nSmallest element is %d",sml);

printf("\nSets are(Column wise): ");

e=0;

printf("\n");

for(i=0;i<d;i++)

{

for(j=0;j<poss/g[i];j++)

{

for(k=1;k<=g[i];k++)

printf("%d",k);

}

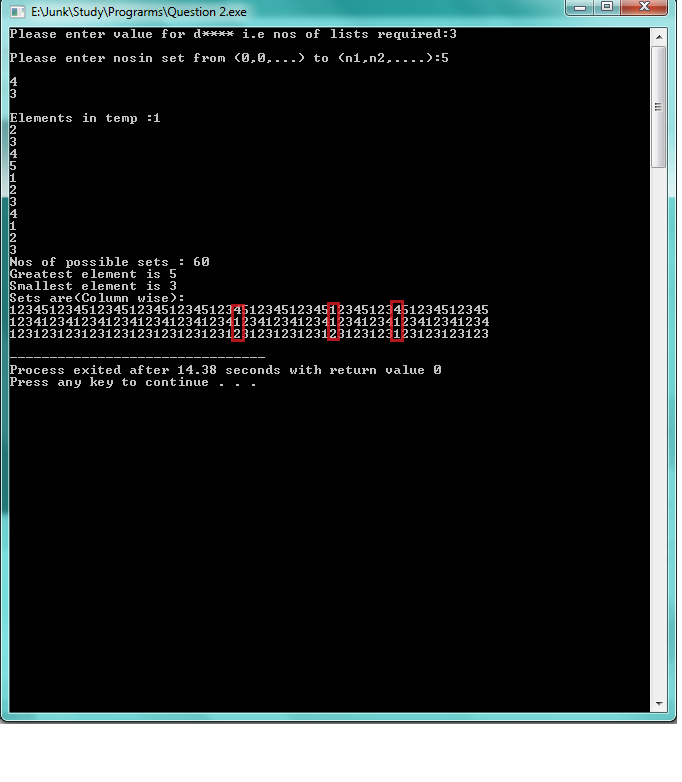
printf("\n");

}

return 0;

}

**OUTPUT:**



Here red boxes represent set of elements, so all the possible sets are mapped VERTICALLY .Each of vertical column represents a set.

**The recursive algorithm for this is below :**

Let g( ) be list of numbers

g( )= { n1,n2,n3,……nd }

// defining the g( ) function and assigning numbers in it.

initial(i)

{

if(i<0)

return;

for ( int j=0,j<g(i),j++)

{

do statement ;

}initial(i-1);

}

Here if i < 0 then it returns to the function and goes into the for loop which in turn it increment’s by incrementing operator  j++ and then it returns to the statement goes in recursive loop

First run: initial(d-1)

initial (i,j)

{

if(j==0)

{

if (i==0)

return;

initial (i-1,g(i-1));

}

else

{

do statement;

initial (I,j-1);

}

}

Now comes to the next element d-1 , if i is 0 then it returns to the inner loop, if not it goes to else statement and comes out of the loop for further iterations.

initial(d-1,g(0))

initial ()

{

for(int i=0;i<g(0);i++)

{

do statement;

}

// this algorithm is for recursive function.

**Problem 1-3.**

Is log *n* = *O*(*n*) ?

Is log *n* = Ω(*n*) ?

Is log *n* = Θ(*n*) ?

Solution:

a. f(n) is O(g(n)) if there are positive real constants c and n0 such that 0 ≤ f(n) ≤ cg(n)

for all values of n ≥ n0.

f(n) ≤cg(n) for all n≥n0

g(n) is called an asymptotic upper bound of f(n)

We write f(n)=Og(n)

It reads f(n)is big oh of g(n)

log(n)=O(n)

f(n)=log(n)

cg(n)=O(n)

If n=1; then log(1)=0 which is less then O(n)

If n=2; then log(2)=0.301 which is less then O(n)

If n=3; then log(3)=0.77 which is less then O(n)

If n=4; then log(4)=0.602 which is less then O(n)

.

.

.

1g(n)=i\*n

>log(n)

Thus f(n)=O(g(n))

Thus log(n)=O(n) for all n>0 and g(n) is an asymptotic upper bound for f(n)

b. f(n) is Ω(g(n)) if there are positive real constants c and n0 such that 0 ≤ cg(n) ≤ f(n)

for all values of n ≥ n0.

f(n) ≥cg(n) for all n≥n0

g(n) is called an asymptotic lower bound of f(n)

We write f(n)= Ω g(n)

It reads f(n) as omega of g(n)

log(n)=Ω(n)

f(n)=log(n)

cg(n)= Ω(n)

If n=2;then log(2)=0.301 and if constant 'c' of cg(n) is 1/4 then cg(n)<f(n)

if n=3;then log(3)=0.77 and if constant 'c' of cg(n) is 1/4 then cg(n)<f(n)

.

.

g(n)/4=n/4

<log(n)

Thus f(n)= Ω(g(n))

Thus log(n)= Ω(n) for c=1/4 and g(n) is an asymptotic lower bound for f(n)

c. f(n) is Θ(g(n)) if there are positive real constants c1, c2, and n0 such that 0 ≤ c1g(n) ≤ f(n) ≤ c2g(n) for all values of n ≥ n0.

f(n) is Θ(g(n)) if f(n) = O(g(n)) and f(n) = Ω(g(n))

Now since log(n)=O(n) and log(n)= Ω(n), g(n) is an asymptotically tight bound for f(n)

**Problem 1-4.** There are *n* bacteria and 1 virus in a Petri dish. Within the first minute, the virus kills one bacterium and produces another copy of itself, and all of the remaining bacteria reproduce, making 2 viruses and 2. (*n* −1) bacteria. In the second minute, each of the viruses kills a bacterium and produces a new copy of itself (resulting in 4 viruses and *2(2(n-1)-2) = 4n-8* bacteria; again, the remaining bacteria reproduce. This process continues every minute.

Will the viruses eventually kill all the bacteria?

If so, design an algorithm that computes how many steps it will take and submit sample runs showing the output of

your code.

How does the running time of your algorithm depend on n?

Solution: Yes viruses will eventually kill the bacteria's.

It takes same number of steps as number of minutes viruses take to kill bacteria's.

Pseudo code:

n<-number of bacterias

v<-viruses

min<-represents time taken

for i <- 0 to n

v <- 2\*v (Doubles the number of viruses every minute)

temp <- 2\*n-2

n <- min\*temp-min (Bacteria's are incremented based on their constraint i.e (2(2(n-1)-2)

n <- n-v (Kill bacteria's)

min++

Code:

#include<stdlib.h>

#include<stdio.h>

int display(int time);

int main()

{

int n=0,v=1;

int min=1,i=0,temp=0;

printf("Please enter the nos of Bacterias:");

scanf("%d",&n);

printf("A virus kills a bacteria");

//for initial kill

n=n-v;

for(i=0;i<n;i++)

{

v=2\*v;

temp=2\*n-2;

n=min\*temp-min;

n=n-v;

min++;

}

min++;

display(min);

return 0;

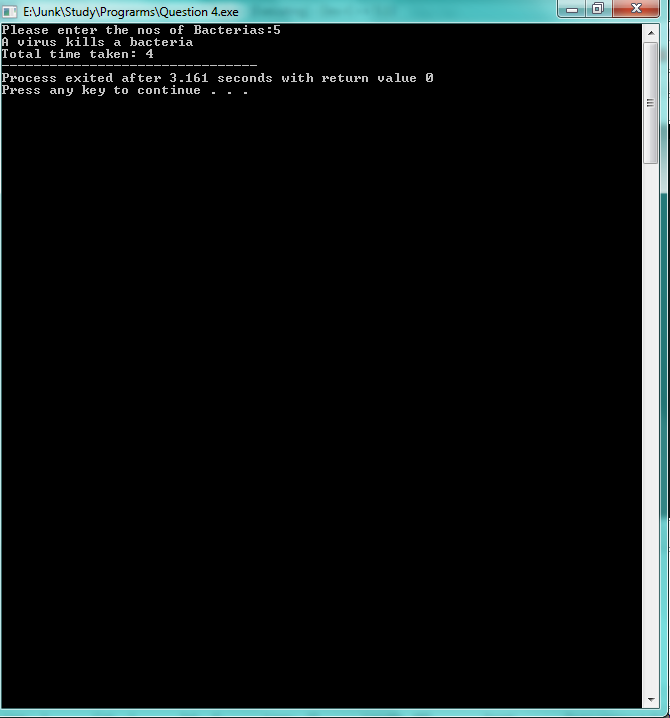
}

int display(int time)

{

printf("\nTotal time taken: %d",time);

}



**Problem 1-5.**

1. Go to the following web page:

http://www.ncbi.nlm.nih.gov/Class/MLACourse/Modules/BLAST/q\_jurassicparkDNA.html

2. Copy the DNA sequence from the book Jurassic Park.

3. Go the NCBI Blast home page at http://www.ncbi.nlm.nih.gov/BLAST/. Go to the link that says

“nucleotide BLAST”.

4. Copy and paste the DinoDNA DNA sequence into the text box, make sure the “nucleotide collection (nr/nt)”

database is selected and hit the “BLAST” button at the bottom.

5. Sort the results by “Total score”.

1. What is the highest score?

2. What is the description of the result with the highest score?

3. What is the query coverage of this result?

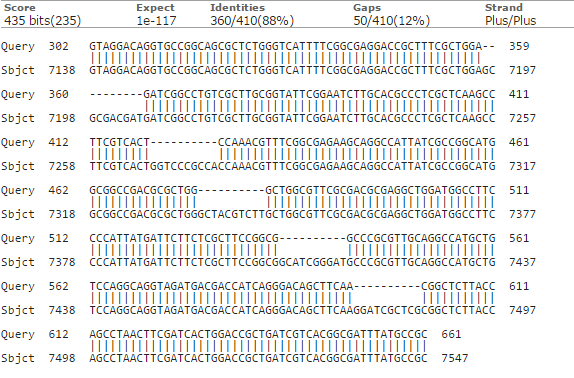
4. Is the DNA sequence in Jurassic park fictional (i.e. made up/random) or “borrowed” (i.e. copied from real

DNA)?

Solution:

5.1. The highest score is 435

5.2. The description is as follow:-



5.3. The query coverage of this result is 98%

5.4. Yes the DNA sequence in Jurassic park is “borrowed” (i.e. copied from real

DNA).

**Problem 1-6.** (20 points)

Write a Matlab program that does something fun. This is your choice entirely -- it can be original or from http://www.mathworks.com/products/bioinfo/ or some other website. You won't be graded on how much fun I find it.

Describe the program in a sentence or two, give a sample input and output, and include a snapshot/screenshot of it in action.

Solution: I have selected the concept of convex hull. In [mathematics](https://en.wikipedia.org/wiki/Mathematics), the convex hull is a set of  X points in the [Euclidean plane](https://en.wikipedia.org/wiki/Euclidean_plane) or [Euclidean space](https://en.wikipedia.org/wiki/Euclidean_space), it is the smallest [convex set](https://en.wikipedia.org/wiki/Convex_set) that contains X. A convex hull can be visualized as the shape enclosed by a rubber band stretched around X. It consists only of outer most points in an Euclidean space.

Code:

% Create the data.

n = 50;

X = randn(n,3);

% Plot the points.

plot3(X(:,1),X(:,2),X(:,3),'ko','markerfacecolor','k');

% Compute the convex hull.

C = convhulln(X);

% Plot the convex hull.

hold on

for i = 1:size(C,1)

j = C(i,[1 2 3 1]);

patch(X(j,1),X(j,2),X(j,3),rand,'FaceAlpha',0.6);

end

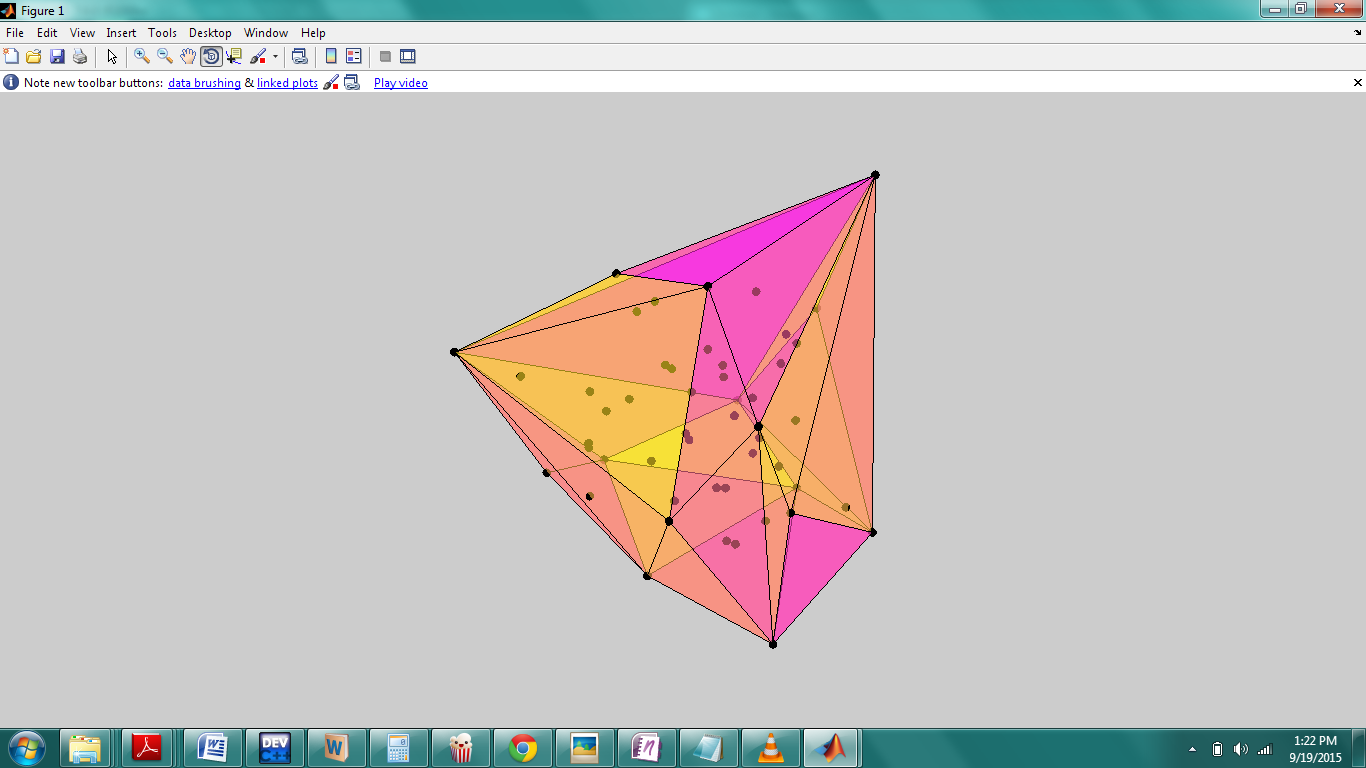
% Modify the view.

view(3), axis equal off tight vis3d; camzoom(1.2)

colormap(spring)

rotate3d on

Output:



References:

[1] Dr. Firas Khatib, "Introduction to Algorithms and complexity" Lecture\_9-8-2015, Chapter 2 (Power point presentation)

[2]Matlab 7.6.0(R2008a), Demos-Matlab/Mathematics/Tessellation and Interpolation of Scattered Data

[3] <http://comments.gmane.org/gmane.comp.programming.algogeeks/1602>