**SHELL SORT**

Name : ROHIT VIJAY

Roll No : R177219150

Contribution : Algorithm Introduction and Complexity Analysis

Name : ROHAN NYATI

Roll No : R177219148

Contribution : Flow chart , Algorithm and Examples with explanation

Name : SAMARTH SUNDRIYAL

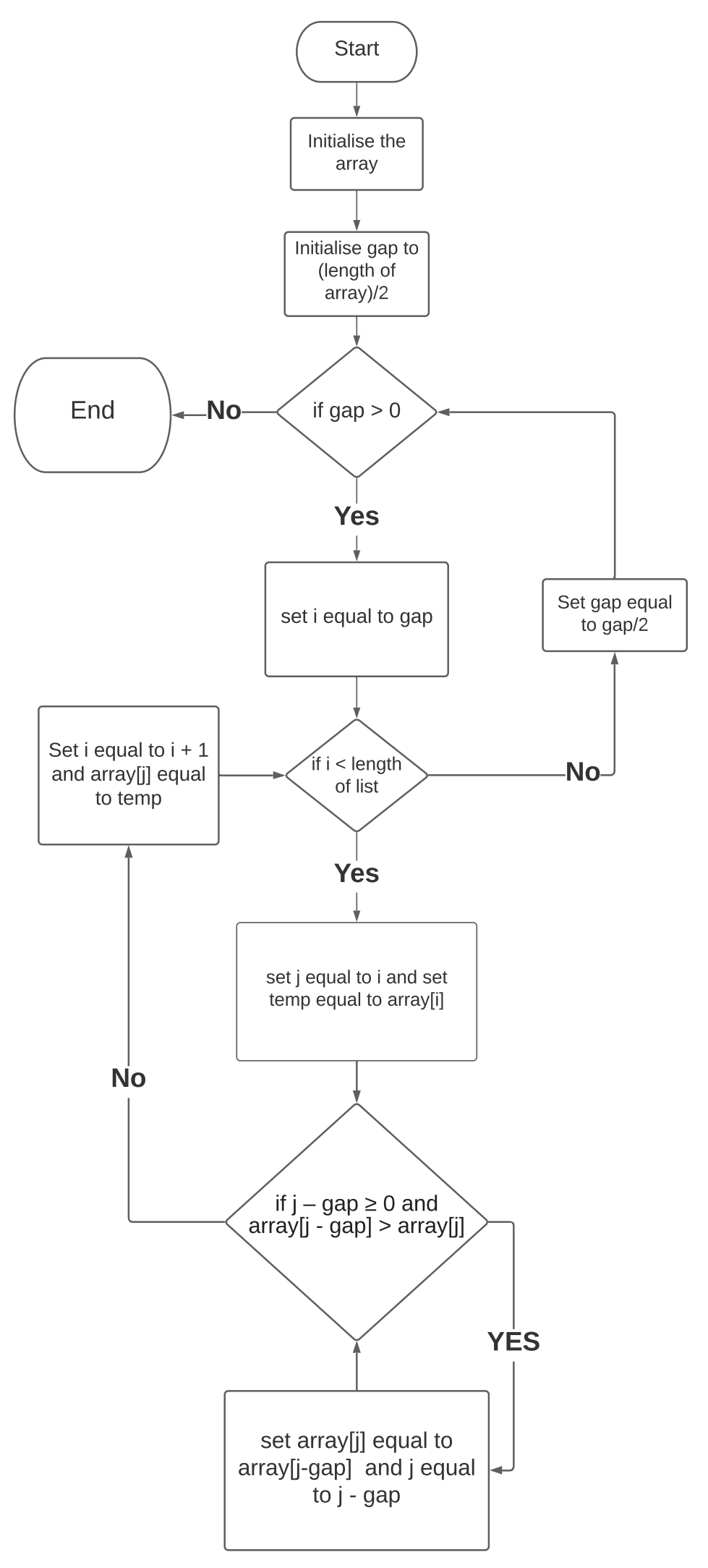
Roll No : R177219160

Contribution : C and Python code

**Algorithm Introduction :**

Shell Sort was Invented by Donald Shell in 1959 . It was the 1st algorithm to break the quadratic time barrier .Shell sort works by comparing elements that are distant rather than adjacent elements in an array , this is why it is an unstable sorting algorithm and this is what makes it different from the Insertion Sort . This algorithm is among the algorithms which are still under research .

**Flowchart :**

****

**Shell Sort Algorithm :**

Begin ShellSort( list )

set gap ←

(1) while gap ≠ 0

set i ← gap

(2) while i < length of list

set j ← i , temp ← list[i]

(3) while j – gap ≥ 0 and list[j - gap] > list[j]

set list[j] ← list[j – gap] , j ← j – gap

end (3)

list[j] ← temp

i ← i + 1

end (2)

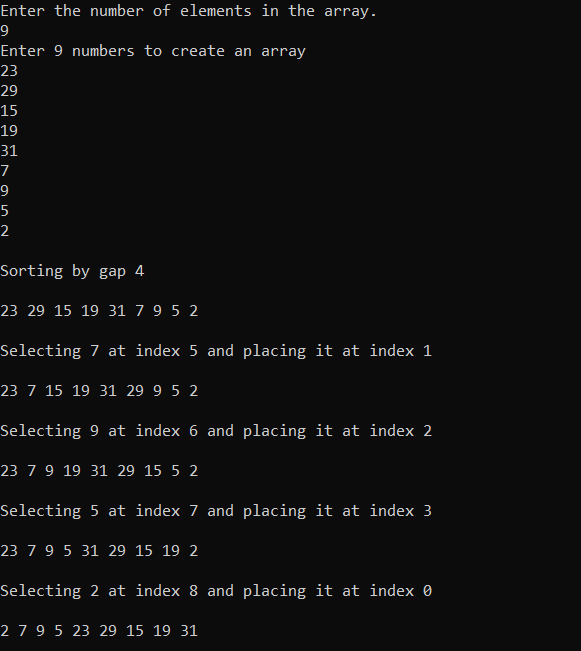
gap ←

end (1)

End ShellSort

**EXAMPLES:**

**1.)Normal case:**



Here, first we give the no. of inputs we have to take for the unsorted array.

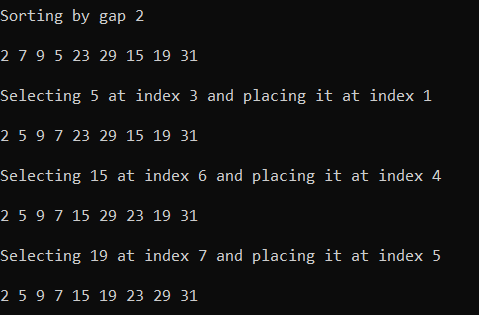
In the given example we have taken no. of values for unsorted array as 9, and then we have given the 9 inputs.

Now we calculate the gap in order of N/2,N/4….till gap value reaches 1(where N= no. of array inputs taken).

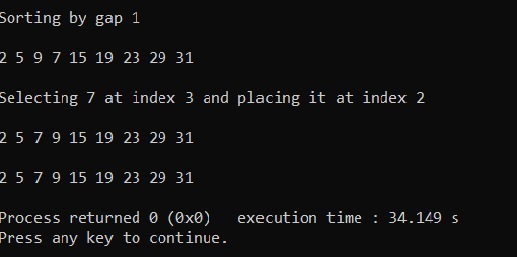
For 1st time when gap value =9/2 =4 ;

Here 1st we compare value at index 0 with value at index 4 since they are in correct order , Therefore we don’t do any swapping. Then we increment index values, now we check for values at index 1 and index 5 , since they don’t follow order ,Therefore now we swap their values , and again increment the index values and keep on checking until “j” value in the loop reaches “n-1”.

Since, we know that in shell sorting we check for values at a difference of gap not only in forward direction but also check and compare them and swap them with the values not in order in backward direction also. Here we can see this happening in the given example after we swap index at 7 and 3 then we compare values at index 4 and 8 , and since they are not in order therefore we swap them but we don’t stop here since we can see that index 4 and index 0 also have a gap of 4 therefore now we check order for values at index 0 and index 4 , and since they are not in order, hence we swap their values as we can see in the last step of loop for the gap of 4.



Now , we follow the same steps for the gap of 2 and finally for the gap of 1 , and at the end we print out the final output of the sorted array using shell sort.



**2.)Best case :**

In best case scenario we know that the array taken as an input is already sorted.

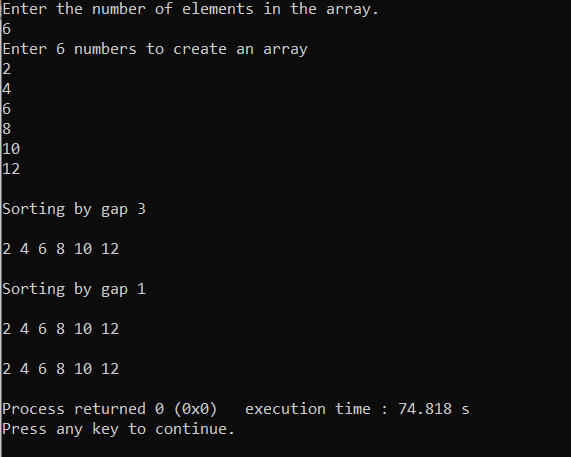
Here in the given example we have taken 6 inputs from the user .

So first we calculate the gap in order of N/2,N/4….till gap value reaches 1(where N= no. of array inputs taken).

For 1st time when gap value =6/2 =3 ;

Then we check the values at index 0 and compare it with the value at index 3 and check whether they are sorted or not if not then we swap their values and in the same way we keep on incrementing the values of “i” and “j” in the loops and check the values at index with difference of 3,whether they are in sorted order or not until “j” value in loop reaches to “n-1”.

Now for 2nd time we get the gap value = 3/2 = 1, here we follow the same steps ; and since here the array is already sorted therefore we don’t see any swapping of values at any index and at last we print out the sorted array.



**Complexity Analysis :**

Shell sort function works on the basis of gap sequence , the most general gap sequence that we take to calculate the gap is N/2,N/4….till gap value reaches 1(where N= no. of array inputs taken). Therefore if we follow this gap sequence we can say that the performance for the given shell sort as:-

**Performance**

Worst-case time complexity: always <=

Best-case time complexity: :

Average-case time complexity:

Space complexity:

Now , since we know that shell sort is a research algorithm and therefore we cannot calculate its actual complexity using any particular methods(since it is still under research).

**Advantages**

Shell sort is an efficient method for medium size lists . It is much faster than bubble sort and insertion sort . It is much preferred than insertion sort when we have an unsorted array and numbers with lesser values lie to the right of the array.

**Disadvantages**

Shell sort has a complex algorithm and it is not as much efficient as compared to merge , heap and quick sort.The running time of Shell sort depends on the choice of increment sequence. For bigger size lists Shell sort is not an efficient method .

**Shell Sort Code :**

**1.)C language**

#include<stdio.h>

#include<stdlib.h>

#include<stdbool.h>

#include<time.h>

//Generates an array of random numbers of a given length. Each number is less than 100.

int\* random\_array\_generator ( int len )

{

srand(time(NULL));

int\* array = (int\*) malloc(sizeof(int) \* len);

while(len--)

array[len] = rand()%100;

return array;

}

int\* input\_array( int len )

{

int\* array = (int\*) malloc(sizeof(int) \* len);

printf("Enter %d numbers to create an array\n", len);

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

scanf("%d",array + i );

return array;

}

//Prints an array of given length.

void print\_array( int\* array , int len )

{

printf("\n");

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

printf("%d ",array[i]);

printf("\n");

}

void shell\_sort( int \*array , int len )

{

int gap = len/2;

while(gap > 0)

{

printf("\nSorting by gap %d\n",gap);

print\_array(array , len);

for( int j , i = gap ; i < len ; i++ )

{

j = i;

int temp = array[i];

while( j - gap >= 0 && array[j - gap] > temp )

{

array[j] = array[j-gap] ;

j -= gap;

}

if(i!=j)

{

array[j] = temp;

printf("\nSelecting %d at index %d and placing it at index %d\n",temp , i , j);

print\_array(array,len);

}

}

gap /= 2;

}

}

int main()

{

//Inputting the length of array.

printf("Enter the number of elements in the array.\n");

int len ;

scanf("%d",&len);

int \*array;

//Generating a random array.

//array = random\_array\_generator( len );

//Inputting array from user.

array = input\_array( len );

//Print the unsorted array.

//print\_array( array , len );

//Sorting the array.

shell\_sort( array , len );

//Print the sorted array.

print\_array( array , len );

//Deallocating the memory.

free(array);

}

**2.)Python**

import random

def random\_array\_generator( length ):

""" Generates a list of random numbers """

numbers = [ ]

for i in range(length):

random\_number = random.randint(0,100)

numbers.append(random\_number)

return numbers

def input\_array( length ):

print("Enter", length ,"numbers to create an array");

return [int(input()) for \_ in range(length)]

def shell\_sort( numbers ):

""" Perform shell sort on a list of numbers """

gap = len(numbers)//2

while( gap > 0 ):

i = gap

while ( i < len(numbers) ):

j = i

temp = numbers[i]

while( j - gap >= 0 and numbers[j - gap] > temp ):

numbers[j] = numbers[j - gap]

j-=gap

numbers[j] = temp

i+=1

gap = gap//2

return numbers

if \_\_name\_\_ == "\_\_main\_\_":

length = int(input("Enter the length of the array\n"))

#unsorted\_numbers = random\_array\_generator(length)

unsorted\_numbers = input\_array(length)

print(unsorted\_numbers)

sorted\_numbers = shell\_sort( unsorted\_numbers )

print(sorted\_numbers)