Project

By: Milton Blaesild 910508-4173
Email: Miletow@gmail.com

I have implemented the following algorithms:

cocktail sort **Difficulty: easy**
Sources used: http://en.wikipedia.org/wiki/Cocktail_sort

gnome sort **Difficulty: easy**
Sources used: http://en.wikipedia.org/wiki/Gnome_sort

shell sort **Difficulty: moderate**
sources used: http://en.wikipedia.org/wiki/Shell_sort
https://www.youtube.com/watch?v=ddeLSDsYVp8&t=173s*/

comb sort **Difficulty: moderate**
sources used: http://en.wikipedia.org/wiki/Comb_sort

cycle sort **Difficulty: moderate**
sources used: http://en.wikipedia.org/wiki/Cycle_sort

quicksort **Difficulty: hard**
sources used: http://en.wikipedia.org/wiki/Quicksort

heapsort **Difficulty: hard**
sources used: http://en.wikipedia.org/wiki/Heapsort
https://www.youtube.com/watch?v=MtQL_ll5KhQ
https://www.youtube.com/watch?v=2DmK_H7IdTo&t=135s

Supplementary code:

Function to check if sorted correctly

```
void CheckSort(int a[], int n){

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

				if(a[ii]>a[ii+1])		//Compare
printf("\nALERT!! %d is bigger than %d on spot %d and %d", a[ii], a[ii+1], ii, ii+1);

		}

		printf("\nChecksort Done!");

}
```

Function to print array

```
void print(int data[], int n){

printf("\n");
for (int ii = 0; ii < n; ++ii) {
printf ("%d\n", data[ii]);   }

}
```

Generate random numbers

```
int n;
scanf("%d", &n);

int randomNumbs[n];

for(int i = 0; i<n; i++){
		randomNumbs[i] = rand();
		}

for(int i = 0; i<n; i++){
		printf("%d\n", randomNumbs[i]);
		}
```

Cocktail sort

Code:

```
void CockTail(int data[], int n){

int tmp;

int StartIx, EndIx;
StartIx = 0;
EndIx = n-1;
int sw = 1;

while(sw == 1){

		sw = 0;
			print(data, 8);			//Print array
		for (int i = StartIx; i < EndIx; ++i) {		//Loop left to right
			if (data[i] > data[i + 1]) {
			tmp = data[i];
			data[i] = data[i + 1];
			data[i + 1] = tmp;
			sw = 1;


			}
		}
			print(data, 8);			//Print array
		if(sw == 0)					//Break if no switch
			break;

		sw = 0;

		--EndIx;

		for (int i = EndIx -1; i >= StartIx; --i) {		//Loop right to left
			if (data[i] > data[i+1]) {
			tmp = data[i];
			data[i] = data[i + 1];
			data[i + 1] = tmp;
			sw = 1;


			}
		}
		++StartIx;

}}
```

Output:

```
Cocktail                                    Done
          0        -3        -3        -3        -3
         -3         0         0         0         0
         10         3         1         1         1
          3         1         3         1         1
          1         4         1         3         3
          4         7         4         4         4
          7         1         7         7         7
          1        10        10        10        10
```

Checksort Done! No alerts. It passes the test of 100000 random integers.

Gnome sort

Code:

```
void gnomeSort(int data[], int n){

int first = 1;

while(first<n)
{
        if(data[first - 1] <= data[first])      //If Correct order move along
        {
                first++;
        }
        else{                                   //Else switch place
                int tmp = data[first-1];
                data[first-1] = data[first];
                data[first] = tmp;
                print(data, n);                 //Print array
                if(-- first == 0)               //Keep moving back until first if satisfied
                {
                        first = 1;
                }
        }

}

}
```

Output:

Gnome Done

-3	-3	-3	-3	-3	-3	-3	-3	-3	-3	-3
0	0	0	0	0	0	0	0	0	0	0
10	3	3	1	1	1	1	1	1	1	1
3	10	1	3	3	3	3	3	3	1	1
1	1	10	10	4	4	4	4	1	3	3
4	4	4	4	10	7	7	1	4	4	4
7	7	7	7	7	10	1	7	7	7	7
1	1	1	1	1	1	10	10	10	10	10

Checksort Done! No alerts. It passes the test of 100000 random integers.

Shell sort

Code:

```
void shellSort(int data[], int n)

{
// Start with a big gap, then reduce the gap
for (int gap = n/2; gap > 0; gap /= 2)                          //Create gap
{

for (int i = gap; i < n; i += 1)
                {
                int tmp = data[i];              //Store value

                print(data, 8);                 // Print array

                int j;
                for (j = i; j >= gap && data[j - gap] > tmp; j -= gap)    //Move until tmp is bigger
                data[j] = data[j - gap];


                data[j] = tmp;      //  put tmp in its correct location



                }
}

}
```

Output:

First gap is 4, second is 2 and last is 1.

Shellsort	Gap 4	Gap 4	Gap 4
0	0	0	0
-3	-3	-3	-3
10	10	10	7
3	3	3	3
1	1	1	1
4	4	4	4
7	7	7	10
1	1	1	1

Gap 2	Gap 2	Gap 2	Gap 2	Gap 2	Gap 2
0	0	0	0	0	0
-3	-3	-3	-3	-3	-3
7	7	7	1	1	1
1	1	1	1	1	1
1	1	1	7	7	7
4	4	4	4	4	4
10	10	10	10	10	10
3	3	3	3	3	3

Gap1	Gap1	Gap1	Gap1	Gap1	Gap1	Gap1	Done
0	-3	-3	-3	-3	-3	-3	-3
-3	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
7	7	7	7	7	3	3	3
3	3	3	3	3	7	7	4
10	10	10	10	10	10	10	7
4	4	4	4	4	4	4	10

Checksort Done! No alerts. It passes the test of 100000 random integers.

Comb sort

Code:

```
int getNextGap(int gap)
{
// Shrink gap by Shrink factor
gap = (gap*10)/13;
if (gap < 1)
return 1;
return gap;
}

void combSort(int data[], int n ){

int gap = n;
int shrink = 1.3;
int sorted = 0;

while(sorted == 0){
                gap = getNextGap(gap);
                if(gap > 1)
                                sorted = 0;
                else
                                sorted =1;

                for(int i = 0; i+gap<n; i++){
                                if(data[i]> data[i+gap]){
                                                int tmp = data[i];
                                                data[i] = data[i+gap];
                                                data[i+gap] = tmp;
                                                sorted = 0;
                                                print(data, 8);                 //Print array
                                }
                }
}}
```

Output:

This shows whenever a gap makes a switch. Gap one makes two switches, gap two one etc. Yellow mark shows which numbers gets switched.

Comb

Gap1	Gap1	Gap2	Gap3	Gap3	Gap4	Gap4		Done
0	0	0	0	0	0	-3	-3	-3
-3	-3	-3	-3	-3	-3	0	0	0
10	7	7	4	1	1	1	1	1
3	3	1	1	1	1	1	1	1
1	1	1	1	4	4	4	3	3
4	4	4	7	7	3	3	4	4
7	10	10	10	10	10	10	10	7
1	1	3	3	3	7	7	7	10

Checksort Done! No alerts. It passes the test of 100000 random integers.

Cycle sort

Code:

```
void cycleSort(int arr[], int n)
{
int tmp;
int writes = 0;

for (int cycle_start = 0; cycle_start <= n - 2; cycle_start++) {
int item = arr[cycle_start];

int pos = cycle_start;
for (int i = cycle_start + 1; i < n; i++)          //Count all smaller elements on the right side
                if (arr[i] < item)
                                pos++;

if (pos == cycle_start)                            // If item is already in correct position continue
                continue;

while (item == arr[pos])                           // ignore all duplicate  elements
                pos += 1;

if (pos != cycle_start) {                          // put the item in it's  correct position

                tmp = arr[pos];
                arr[pos] = item;
                item = tmp;
                writes++;
}

while (pos != cycle_start) {                                    // Rotate rest of the cycle
                pos = cycle_start;

                for (int i = cycle_start + 1; i < n; i++)
                                if (arr[i] < item)
                                pos += 1;

                while (item == arr[pos])                        // ignore all duplicate elements
                                pos += 1;

                if (item != arr[pos]) {                         // put the item in it's correct position

                                tmp = arr[pos];
                                arr[pos] = item;
                                item = tmp;

                                writes++;
                }
}
print(arr, 8);                                                  //Print array
} }
```

Output:

CycleSort			Done
0	-3	-3	-3
-3	0	0	0
10	10	1	1
3	3	3	1
1	1	1	3
4	4	4	4
7	7	7	7
1	1	10	10

Checksort Done! No alerts. It passes the test of 100000 random integers.

Quick sort

Code:

```
int partition(int A[], int lo, int hi){
int pivot, j , i, tmp;

                //Furthest to the right
                pivot = A[hi];
                i = lo;

                for(j = lo; j<hi; j++){
                                if(A[j]<pivot){
                                                tmp = A[i];
                                                A[i] = A[j];
                                                A[j] = tmp;
                                                i++;
                                // i Increases when shift occur. I is the split(pivot) for next recursion.
                                }
                }
                tmp = A[i];
                A[i] = A[hi];
                A[hi] = tmp;

                print(A, 8);
                return i;
                }


void quicksort(int A[], int lo, int hi){
if (lo < hi){
                int p = partition(A, lo, hi);
                quicksort(A, lo, p-1);
                quicksort(A, p+1, hi);

}               }
```

Output:

Column 1 is first quicksort that recursively leads to 1.1 and 1.2 which leads to 1.1.1 etc.

Quicksort

1	1.1	1.2	1.1.1	1.1.2	1.2.1	1.2.2	Result
0	0	-3	-3	-3	-3	-3	-3
-3	-3	0	0	0	0	0	0
10	1	1	1	1	1	1	1
3	3	3	3	3	3	1	1
1	1	1	1	1	1	3	3
4	4	4	4	4	4	4	4
7	7	7	7	7	7	7	7
1	10	10	10	10	10	10	10

Checksort Done! No alerts. It passes the test of 100000 random integers.

Heapsort

Code:

```
int leftChild(int i){ int d = 2*i +1;
return d;}
int rightChild(int i){ int d = 2*i +2;
return d;}
int iParent(int i){ double d = floor((i-1)/2);                    // #include <math.h>
return d;}

void siftDown(int a[], int start, int end){


int tmp;
int root = start;

while(leftChild(root)<= end){

                int child = leftChild(root);
                int swap = root;
//Swap if child is bigger than root
                if(a[swap] < a[child]){
                swap = child;
                }
//Swap if child is bigger than root
//If child one was bigger it now compares to that making the largest into root.
                if(child+1 <= end && a[swap] < a[child+1]){
                                swap = child +1;
                }
                if (swap == root){
                                break;                          //Break if no switch
                }
                else{

                tmp = a[root];
                a[root] = a[swap];
                a[swap] = tmp;
                root = swap;                                    //Make child into root and move on
                }
}}

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

int start = iParent(n-1);                       //Start from lowest parent
while(start>= 0){
                siftDown(a, start, n-1);        //Put in heap order
                start = start-1;                // Move up in heap
}
print(a, 8);                                    //Print max-heap
}
```

```
void swap(int a[], int i, int j){

int tmp;
                tmp = a[i];
                a[i] = a[j];
                a[j] = tmp;
}

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

heapify(a, n);                          //Create max-heap

int end = n-1;
while(end>0){

                swap(a, end, 0);        //Swap sorted element

                --end;                  //Reduce sorting span

                siftDown(a, 0, end);    //Re-heap
                print(a, 8);            //Print array

                }
}
```

Output:

The yellow mark shows the max number in the heap which gets put at the current end position, represented by a green mark.

Original	Max-Heap								Done
0	10	7	4	3	1	1	0	-3	-3
-3	3	3	3	1	1	-3	-3	0	0
10	7	4	0	0	0	0	1	1	1
3	1	1	1	-3	-3	1	1	1	1
1	1	1	1	1	3	3	3	3	3
4	4	-3	-3	4	4	4	4	4	4
7	0	0	7	7	7	7	7	7	7
1	-3	10	10	10	10	10	10	10	10

Checksort Done! No alerts. It passes the test of 100000 random integers.