

## Source Code

```

1  #include<stdio.h>
2  #include<stdlib.h>
3  #include<string.h>
4  #include<time.h>
5
6  typedef struct DATAstruct{ //declare array to keep data in 3 fields
7      long col;
8      long long f1;
9      char f2[100], f3[100];
10 } DATAstruct; //declare in name DATAstruct
11
12 void readfile(DATAstruct data[], int *count){ //declare function to read file .csv
13     FILE *fp;
14     DATAstruct x;
15     if((fp = fopen("test.csv","r")) != NULL){ //if file was found
16         while(fscanf(fp, "%lu, %llu, %[^,], %s", &x.col, &x.f1, &x.f2, &x.f3) == 4){ //then receive in array each field
17             data[*count] = x; //and count number of data
18             *count = *count +1;
19         }
20         fclose(fp);
21     }else {
22         printf("\nCannot read file.\n"); //if file was not found then show warning
23     }
24 }
25
26 void swap(DATAstruct *a, DATAstruct *b){ //function to swap data
27     DATAstruct c;
28     c = *a;
29     *a = *b;
30     *b = c;
31 }
32
33 //function of selection sort
34 void sltsort(DATAstruct data[], int start, int stop){
35     int i, j, min;
36     for(i = start; i < stop; i++){
37         min = i; //give first location is minimum
38         for(j = i + 1; j <= stop; j++){
39             if(data[j].f1 < data[min].f1) { //if next location is less than minimum
40                 min = j;
41             }
42         }
43         swap(&data[min], &data[i]); //then swap data in location
44     }
45 }
46
47 //function to count time in selection sort
48 double time_slt(DATAstruct data[], DATAstruct test[], int count){
49     clock_t start, end;
50     int i, j;
51     double time;
52     for(i = 0; i < count; i++){ //copy data in array data to array test
53         test[i] = data[i]; //for data in array data will not change
54     }
55     start = clock();
56     sltsort(test, 0, count);
57     end = clock();
58     time = (double)(end - start) / CLOCKS_PER_SEC;
59     return time;
60 }
61
62 //function for insertion sort
63 void istsort(DATAstruct data[], int start, int stop){
64     int i, j, x;
65     for(i = start + 1; i <= stop; i++){
66         x = data[i].f1;
67         for(j = i; ((j > start) && (x < data[j-1].f1)); j--){ //loop to find data that less than x then swap
68             data[j].f1 = data[j-1].f1; //exchange data in every location by location-1
69             data[j-1].f1 = x;
70         }
71     }
72 }
73
74 //function to count time in insertion sort
75 double time_ist(DATAstruct data[], DATAstruct test[], int count){
76     clock_t start, end;
77     int i, j;
78     double time;
79     for(i = 0; i < count; i++){
80

```

```

81     test[i] = data[i];
82 } //copy data in array data to array test
83 //for data in array data will not change
84
85     start = clock();
86     istsort(test, 0, count-1);
87     end = clock();
88     time = (double)(end - start) / CLOCKS_PER_SEC;
89     return time;
90 }
91
92 //function for bubble sort
93 void bbsort(DATAstruct data[], int start, int stop){
94     int i, j;
95     for(i = start; i <= (stop-1); i++){
96         for(j = stop; j > i; j--){
97             if(data[j].f1 < data[j-1].f1){ //compare data
98                 swap(&data[j], &data[j-1]); //and swap
99             }
100         }
101     }
102 }
103
104 //function to count time in bubble sort
105 double time_bb(DATAstruct data[], DATAstruct test[], int count){
106     clock_t start, end;
107     int i, j;
108     double time;
109     for(i = 0; i < count; i++){
110         //copy data in array data to array test
111         //for data in array data will not change
112         test[i] = data[i];
113     }
114     start = clock();
115     bbsort(test, 0, count-1);
116     end = clock();
117     time = (double)(end - start) / CLOCKS_PER_SEC;
118     return time;
119 }
120
121 //function for shell sort
122 void shsort(DATAstruct data[], int start, int stop){
123     int gap, changed, i;
124     gap = stop - start + 1;
125     do{
126         gap = gap / 2; //fix the gap between data
127         do{
128             changed = 0; //changed is for checking if data is swap
129             for(i = start; i < stop - gap + 1; i++){
130                 if(data[i].f1 > data[i+gap].f1){
131                     swap(&data[i], &data[i+gap]);
132                     changed = 1; //if there's changed it's mean sorting is not complete
133                 }
134             }
135         } while(changed == 1);
136     } while(gap > 1); //loop until there will be one group
137 }
138
139 //function to count time in shell sort
140 double time_sh(DATAstruct data[], DATAstruct test[], int count){
141     clock_t start, end;
142     int i, j;
143     double time;
144     for(i = 0; i < count; i++){
145         //copy data in array data to array test
146         //for data in array data will not change
147         test[i] = data[i];
148     }
149     start = clock();
150     shsort(test, 0, count-1);
151     end = clock();
152     time = (double)(end - start) / CLOCKS_PER_SEC;
153     return time;
154 }
155
156 //function to descending sort by selection sort
157 void dessltsort(DATAstruct data[], int start, int stop){
158     int i, j, min;
159     for(i = start; i < stop; i++){

```

```

161     min = i;
162     for(j = i + 1; j <= stop; j++){
163         if(data[j].f1 > data[min].f1) { //sort from max to min
164             min = j;
165         }
166     }
167     swap(&data[min] , &data[i]);
168 }
169 }
170
171 //function to count time in descending sort of selection sort
172 double destime_slr(DATAstruct data[], DATAstruct test[], int count){
173     clock_t start, end;
174     int i, j;
175     double time;
176     for(i = 0; i < count; i++){
177         test[i] = data[i];
178         //copy data in array data to array test
179         //for data in array data will not change
180     }
181     start = clock();
182     desltsort(test, 0, count);
183     end = clock();
184     time = (double)(end - start) / CLOCKS_PER_SEC;
185     return time;
186 }
187
188 //function for descending sort by insertion sort
189 void desistsort(DATAstruct data[],int start, int stop){
190     int i, j, x;
191     for(i = start+1; i <= stop; i++){
192         x = data[i].f1;
193         for(j = i; ((j > start)&&(x > data[j-1].f1)); j--){ //sorting from max to min
194             data[j].f1 = data[j-1].f1; //exchange data in each location
195             data[j-1].f1 = x;
196         }
197     }
198 }
199
200 //function to count time in descending sort of insertion sort
201 double destime_ist(DATAstruct data[], DATAstruct test[], int count){
202     clock_t start, end;
203     int i, j;
204     double time;
205     for(i = 0; i < count; i++){
206         //copy data in array data to array test
207         //for data in array data will not change
208         test[i] = data[i];
209     }
210     start = clock();
211     desistsort(test, 0, count-1);
212     end = clock();
213     time = (double)(end - start) / CLOCKS_PER_SEC;
214     return time;
215 }
216
217 //function for descending sort by bubble sort
218 void desbbsort(DATAstruct data[], int start, int stop){
219     int i, j;
220     for(i = start; i <= (stop-1); i++){
221         for(j = stop; j > i; j--){
222             if(data[j].f1 > data[j-1].f1){ //if data in next location is more than before
223                 swap(&data[j], &data[j-1]); //then swap data
224             }
225         }
226     }
227 }
228
229 //function to count time in descending sort of bubble sort
230 double destime_bb(DATAstruct data[], DATAstruct test[], int count){
231     clock_t start, end;
232     int i, j;
233     double time;
234     for(i = 0; i < count; i++){
235         //copy data in array data to array test
236         //for data in array data will not change
237         test[i] = data[i];
238     }
239     start = clock();
240     desbbsort(test, 0, count-1);

```

```

241     end = clock();
242     time = (double)(end - start) / CLOCKS_PER_SEC;
243     return time;
244 }
245
246 //function for descending sort by shell sort
247 void deshsort(DATAstruct data[], int start, int stop){
248     int gap, changed, i;
249     gap = stop - start + 1;
250     do{
251         gap = gap / 2; //fixed the gap between data
252         do{
253             changed = 0; //changed is for checking if data is swap
254             for(i = start; i < stop - gap + 1; i++){
255                 if(data[i].f1 < data[i+gap].f1 ){
256                     swap(&data[i], &data[i+gap]);
257                     changed = 1;
258                     //if there's changed it's mean sorting is not complete
259                     // then sort until it have no changed
260                 }
261             }
262         } while(changed == 1);
263     }while(gap > 1);
264 }
265
266 //function to count time in descending sort of shell sort
267 double destime_sh(DATAstruct data[], DATAstruct test[], int count){
268     clock_t start, end;
269     int i, j;
270     double time;
271     for(i = 0; i < count; i++){
272         //copy data in array data to array test
273         //for data in array data will not change
274         test[i] = data[i];
275     }
276     start = clock();
277     deshsort(test, 0, count-1);
278     end = clock();
279     time = (double)(end - start) / CLOCKS_PER_SEC;
280     return time;
281 }
282
283 int main(){
284     int count = 0;
285     DATAstruct *data, *test;
286     double time1, time2, time3, time4, time5, time6, time7, time8;
287     double time51, time61, time71, time81;
288
289     data = (DATAstruct *)malloc(100000*sizeof(DATAstruct));
290     test = (DATAstruct *)malloc(100000*sizeof(DATAstruct));
291     //declare 2 arrays because array data will not change data inside array
292
293     readfile(data, &count); //order program to read file and count number of data
294     printf("Loading...\n -> %d records\n", count); //show number of data
295
296     //first round of sorting is normal sorting -- from min to max
297     printf("\nFirst Round of Sorting Progress...\n");
298     time1 = time_slr(data, test, 10000);
299     time2 = time_ist(data, test, 10000);
300     time3 = time_bb(data, test, 10000);
301     time4 = time_sh(data, test, 10000);
302
303     //I order to print data to check if it's sorting correctly
304     printf("\n%llu %s %s\n", test[0].f1, test[0].f2, test[0].f3);
305     printf("\n%llu %s %s\n", test[4999].f1, test[4999].f2, test[4999].f3);
306     printf("\n%llu %s %s\n", test[9999].f1, test[9999].f2, test[9999].f3);
307
308     //second round is sorting by add 1 data from data to sorting together
309     printf("\n\nAdd 1 data");
310     printf("\nSecond Round of Sorting Progress...\n\n");
311     time5 = time_slr(data, test, 10001);
312     time6 = time_ist(data, test, 10001);
313     time7 = time_bb(data, test, 10001);
314     time8 = time_sh(data, test, 10001);
315
316     //I order to print data to check if it's sorting correctly as before
317     printf("\n%llu %s %s\n", test[0].f1, test[0].f2, test[0].f3);
318     printf("\n%llu %s %s\n", test[4999].f1, test[4999].f2, test[4999].f3);
319     printf("\n%llu %s %s\n", test[9999].f1, test[9999].f2, test[9999].f3);
320 }

```

```

321 //third round is descending sort -- sort from max to min
322 printf("\n\nDescending");
323 printf("\nThird Round of Sorting Progress..\n\n");
324 time51 = time_sl(data, test, 10001);
325 time61 = time_is(data, test, 10001);
326 time71 = time_bb(data, test, 10001);
327 time81 = time_sh(data, test, 10001);
328
329 //I order to print data to check if it's sorting correctly as i did before
330 printf("\n%llu %s %s\n",test[0].f1, test[0].f2, test[0].f3);
331 printf("\n%llu %s %s\n",test[4999].f1, test[4999].f2, test[4999].f3);
332 printf("\n%llu %s %s\n",test[9999].f1, test[9999].f2, test[9999].f3);
333
334 //then show table of time that program use to sort in each solution and each round
335 printf("\n\n- Sort - Random data - Insert data - Descending -\n");
336 printf("- Selection - %.3lf s - %.3lf s - %.3lf s -\n",time1, time5, time51);
337 printf("- Insertion - %.3lf s - %.3lf s - %.3lf s -\n",time2, time6, time61);
338 printf("- Bubble - %.3lf s - %.3lf s - %.3lf s -\n",time3, time7, time71);
339 printf("- Shell - %.3lf s - %.3lf s - %.3lf s -\n",time4, time8, time81);
340
341 printf("\nEnd Program\n");
342 printf("\nProgram written by Onwipa Kujaroenpaisan 62070501064"); //show my name at the last line :)
343 return 0;
344 }

```

อธิบายโค้ดคร่าวๆ คือการเปิดและอ่านไฟล์ .csv แล้วแบ่งข้อมูลที่ได้รับออกเป็น 3 ฟิลด์ และนำฟิลด์ที่1 (ซึ่งเป็นตัวเลข) จำนวน 10,000 ตัว มาเรียงด้วยจากน้อยไปมากวิธีการทั้ง 4 แบบ ได้แก่ Selection sort, Insertion sort, Bubble sort และ Shell sort โดยจับเวลาที่ใช้ในการเรียงแต่ละครั้งไว้ด้วย จากนั้นใช้ฟิลด์ตัวเลขต้นฉบับ (ที่ยังไม่ได้เรียง) จำนวน 10,001 ตัว (เพิ่มจากเดิม 1 ตัว) มาเรียงซ้ำอีกครั้ง หลังจากเรียงอีกครั้งแล้วก็นำ 10,001 ตัวที่เป็นต้นฉบับ มาเรียงอีกครั้ง แต่ครั้งสุดท้ายจะเรียงจากมากไปน้อย โดยจับเวลาทุกครั้ง และสร้างตารางแสดงและเปรียบเทียบเวลาที่ใช้ในการเรียงลำดับแต่ละครั้ง โดยทั้งหมดนี้จะจบในการรันครั้งเดียว

## Test Case

```

- Sort - Random data - Insert data - Descending -
- Selection - 0.173 s - 0.161 s - 0.177 s -
- Insertion - 0.001 s - 0.000 s - 0.001 s -
- Bubble - 1.304 s - 1.270 s - 1.273 s -
- Shell - 0.013 s - 0.012 s - 0.012 s -

End Program

Program written by Onwipa Kujaroenpaisan 62070501064

```

ตารางด้านบนนี้คือ เวลาที่ใช้ในการเรียงแต่ละครั้ง โดยจะเห็นได้ว่า Insertion sort ใช้เวลาน้อยที่สุด และ Bubble sort ใช้เวลามากที่สุด จึงสรุปได้ว่า Insertion sort เป็นวิธีที่ดีที่สุด

ปัญหาที่พบในการทำ ASSIGNMENT

- เวลาที่ใช้เรียงของวิธี insertion sort น้อยเกินไป //ไม่ทราบสาเหตุ

**Self-Assessment** : 4 ทำงานด้วยตนเอง แก้ไขปัญหาด้วยตนเองได้ แต่บางครั้งยังต้องขอคำแนะนำจากผู้อื่น