

Project Cover Page

This project is a group project with up to four students per team. For each group member, please print first and last name and e-mail address.

- 1.
- 2.
- 3.
- 4.

Please write how each member of the group contributed to the project.

- 1.
- 2.
- 3.
- 4.

Please list all sources: web pages, people, books or any printed material, which you used to prepare a report and implementation of algorithms for the project.

Type of sources:	
People	
Web Material (give URL)	
Printed Material	
Other Sources	

I certify that I have listed all the sources that I used to develop solutions to the submitted project report and code.

- | | | |
|-------------------|------------|------|
| 1. Your signature | Typed Name | Date |
| 2. Your signature | Typed Name | Date |
| 3. Your signature | Typed Name | Date |
| 4. Your signature | Typed Name | Date |

CSCE 221 Programming Assignment 2 (100 points)

Program and Reports due February 17th by 11:59pm

• Objective

In this assignment, you will implement in C++ five sorting algorithms: selection sort, insertion sort, bubble sort, Shell sort, and radix sort. You will test your code using varied input cases, record computational time and the number of comparisons in each sort algorithm, and compare these computational results with the theoretical running time of the algorithms using Big-O asymptotic notation.

• General Guidelines

1. This project can be done in groups of at most four students. Please use the cover sheet at the previous page for your report.
2. The supplementary program is packed in `221-A2-code.tar` which can be downloaded from eCampus. You need to “untar” the file using the following command on a Linux machine:

```
tar xfv 221-A2-code.tar
```

It will create the directory `221-A2-code`.

3. Make sure that your code can be compiled using a C++ compiler running on a Linux machine before submission because your programs will be tested on such a machine. Use Makefile provided in the directory to compile C++ files by typing the following command:

```
make
```

You may clean your directory with this command:

```
make clean
```

4. When you run your program on a Linux machine, use Ctrl+C (press Ctrl with C together) to stop the program. Do NOT use Ctrl+Z, as it just suspends the program and does not kill the process. We do not want to see the department server down because of this assignment.
5. Supplementary reading

(a) Lecture note: [Introduction to Analysis of Algorithms](#)

(b) Lecture note: [Sorting in Linear Time](#)

6. Submission guidelines

(a) Electronic copy of all your code tested on 15 types of input integer sequences (but do not submit your test files), and reports in \LaTeX and PDF format should be in the directory `221-A2-code`. This command typed in the directory `221-A2-code` will create the tar file (`221-A2-code-submit.tar`) for the submission to eCampus:

```
make tar
```

(b) Your program will be run and tested on TA’s input files.

• Skeleton Code: Description and Implementation

1. In this assignment, the sort program reads a sequence of integers either from the screen (standard input) or from a file, and outputs the sorted sequence to the screen (standard output) or to a file. The program can be configured to show total running time and/or total number of comparisons done by the specific sort algorithm.
2. (20 points) Rewrite the provided code using the STL vector instead of the dynamic array A used in the code. Check if your new code compiles.
3. This program does not have a menu but takes arguments from the command line. The code for interface is completed in the skeleton program, so you only have to know how to execute the program using the command line.

The program usage is as follows. *Note that options do not need to be specified in a fixed order. You do not need to list all the options in the command line. **Do not enter brackets** (see Examples).*

Usage:

```
./sort [-a ALGORITHM] [-f INPUTFILE] [-o OUTPUTFILE] [-h] [-d] [-p]
      [-t] [-T] [-c] [-r]
```

Examples of using the options:

```
./sort -h
./sort -a S -f input.txt -o output.txt -d -t -c -p
./sort -a I -t -c
./sort
```

Description of the options:

-a ALGORITHM: Use ALGORITHM to sort.
ALGORITHM is a single character representing an algorithm:
S for selection sort
B for bubble sort
I for insertion sort
H for shell sort
R for radix sort

-f INPUTFILE: Obtain integers from INPUTFILE instead of STDIN

-o OUTPUTFILE: Place output data into OUTPUTFILE instead of STDOUT

-h: Display this help and exit

-d: Display input: unsorted integer sequence

-p: Display output: sorted integer sequence

-t: Display running time of the chosen algorithm in milliseconds using clock()

-T: Display running time of the chosen algorithm in milliseconds using the chrono class

-c: Display number of comparisons (excluding radix sort)

-r: Provide the number of repeated runs for the sort (default=1)

4. **Format of the input data.** The first line of the input contains a number n which is the number of integers to sort. Subsequent n numbers are written one per line which are the numbers to sort. Here is an example of input data where 5 is the number of integers to sort

```
5
7
-8
4
0
-2
```

5. **Format of the output data.** The sorted integers are printed one per line in increasing order. Here is the output corresponding to the above input:

```
-8
-2
0
4
7
```

6. (25 points) Your tasks include implementation of the following five sorting algorithms in corresponding cpp files.

- (a) selection sort in selection-sort.cpp
- (b) insertion sort in insertion-sort.cpp
- (c) bubble sort in bubble-sort.cpp
- (d) Shell sort in shell-sort.cpp

- (e) radix sort in `radix-sort.cpp`
 - i. The radix algorithm should sort unsigned integers in the range from 0 to $(2^{16} - 1)$.
 - ii. The radix sort can be applied to negative numbers using this input modification:
“You can shift input to get non-negative numbers, sort the data, and shift back to the original data”
- 7. (10 points) In order to test the algorithms generate (one by one) input cases of the sizes 10^2 , 10^3 , 10^4 , and 10^5 integers in three different orders
 - (a) random order
 - (b) increasing order
 - (c) decreasing order

HINT: The standard library `<cstdlib>` provides functions `srand()` and `rand()` to generate random numbers.

- 8. (10 points) Modify code (excluding the radix sort) to count the number of **comparisons performed on input integers**. The following tips should help you with determining how many comparisons are performed.
 - (a) You will measure 3 times for each algorithm (use `-r 3`) on each sequence
 - (b) Use a variable (called `num_cmps`) to keep track of the number of comparisons, typically in `if` or loop statements
 - (c) Remember that C++ uses the shortcut rule for evaluating boolean expressions. A way to count comparisons accurately is to use the comma expression. For example,


```
while (i < n && (num_cmps++, a[i] < b))
```

HINT: If you modify `sort.cpp` and run several sorting algorithms subsequently, you have to call `resetNumCmps()` to reset number of comparisons between every two calls to `s->sort()`.

- 9. To time each sorting algorithm, you use the function `clock()` and the C++ class `chrono` (see the options `-t` and `-T`). You need to compare these two timings and discuss it in your report.
 - (a) Here is the example of how to use the function `clock()`. The timing part for `clock()` is completed in the skeleton program.

The example of using `clock()`:

```
#include <ctime>
...
clock_t t1, t2;
t1 = clock(); // start timing
...
/* operations you want to measure the running time */
...
t2 = clock(); // end of timing
double diff = (double) (t2 - t1) * 1000 / CLOCKS_PER_SEC;
cout << "The timing is " << diff << " ms" << endl;
```

- (b) You can find information about the class `chrono` on this website:
<http://www.cplusplus.com/reference/chrono/>

• Report (35 points)

Write a report that includes all following elements in your report.

1. (2 points) A brief description of assignment purpose, assignment description, how to run your programs, what to input and output.
2. (3 points) Explanation of splitting the program into classes and providing a *description and features of C++ object oriented programming and/or generic programming were used in this assignment*.
3. (5 points) **Algorithms.** Briefly describe the features of each of the five sorting algorithms.

4. (5 points) **Theoretical Analysis.** Theoretically analyze the time complexity of the sorting algorithms with input integers in decreasing, random and increasing orders and fill the second table. Fill in the first table with the time complexity of the sorting algorithms when inputting the best case, average case and worst case. Some of the input orders are exactly the best case, average case and worst case of the sorting algorithms. State what input orders correspond to which cases. You should use big-O asymptotic notation when writing the time complexity (running time).

Complexity	best	average	worst
Selection Sort			
Insertion Sort			
Bubble Sort			
Shell Sort			
Radix Sort			

Complexity	inc	ran	dec
Selection Sort			
Insertion Sort			
Bubble Sort			
Shell Sort			
Radix Sort			

inc: increasing order; dec: decreasing order; ran: random order

5. (15 points) **Experiments.**

- (a) Briefly describe the experiments. Present the experimental running times (**RT**) and number of comparisons (**#COMP**) performed on input data using the following tables.

RT	Selection Sort			Insertion Sort			Bubble Sort		
n	inc	ran	dec	inc	ran	dec	inc	ran	dec
100									
10^3									
10^4									
10^5									

RT	Shell Sort			Radix Sort		
n	inc	ran	dec	inc	ran	dec
100						
10^3						
10^4						
10^5						

#COMP	Selection Sort			Insertion Sort		
n	inc	ran	dec	inc	ran	dec
100						
10^3						
10^4						
10^5						

#COMP	Bubble Sort			Shell Sort		
n	inc	ran	dec	inc	ran	dec
100						
10^3						
10^4						
10^5						

inc: increasing order; dec: decreasing order; ran: random order

- (b) For each of the five sort algorithms, graph the running times over the three input cases (inc, ran, dec) versus the input sizes (n); and for each of the first four algorithms graph the numbers of comparisons versus the input sizes, totaling in 9 graphs.
HINT: To get a better view of the plots, *use logarithmic scales* for both x and y axes.
- (c) To compare performance of the sorting algorithms you need to have another 3 graphs to plot the results of all sorts for the running times for *each* of the input cases (inc, ran, dec) separately.
HINT: To get a better view of the plots, *use logarithmic scales* for both x and y axes.
6. (3 points) **Discussion.** Comment on how the experimental results relate to the theoretical analysis and explain any discrepancies you notice. Is your computational results match the theoretical

analysis you learned from class or textbook? Justify your answer. Also compare radix sort running time with the running time of four comparison-based algorithms.

7. (2 points) **Conclusions.** Give your observations and conclusion. For instance, which sorting algorithm seems to perform better on which case? Do the experimental results agree with the theoretical analysis you learned from class or textbook? What factors can affect your experimental results?