Basic Sorts

110191019 YU-TING CHUNG

- **Introduction**

  Use selection sort, insertion sort, bubble sort, and min-max sort to rearrange the English text string in order starting from the first letter a. There are 9 different data, and each run 500 times. Then, print out the number of data, the names of 4 algorithms, and their average CPU time. Finally, we compare 4 algorithms.

- **Approach**

  - Selection sort

```
67  void SelectionSort(char **list,int n){
68
69      int i, j, k;
70      char *temp;
71
72      for(i = 0; i <= N-1; i++){                          //for every A[i]
73          j=i;                                            //Initialize j to i
74          for(k=i+1; k < n; k++){                         //Search for the smallest in A[i+1:n]
75              if(strcmp(list[k], list[j]) < 0){           //Found, remember it in j
76                  j = k;
77              }
78          }
79          temp = list[i];                                 //Swap A[i] and A[j]
80          list[i] = list[j];
81          list[j] = temp;
82      }
83  }
```

  For an array of length N, find the smallest element in the unsorted part of the array and swap it with the current position. This process repeats until the entire array is sorted.

  Time complexity：The outer loop will execute N times, and the inner loop will execute N-1times.

$$\sum_{i=0}^{n-1}(n-i) = \frac{n(n-1)}{2} = \Theta(n^2)$$

Space complexity: $\Theta(1)$

- Insertion sort

```
85  void InsertionSort(char **list,int n){
86      int i, j;
87      char *temp;
88      for (j = 1; j < N; j++){                          // Assume A[1 : j -1] already sorted.
89          temp = list[j];                               // Move A[j] to its proper place.
90          i = j - 1;                                    // Init i to be j -1.
91          while((i >= 0) && (strcmp(temp, list[i]) < 0)){   // Find i such that A[i] ≤ A[j].
92              list[i + 1] = list[i];                    // Move A[i] up by one position.
93              i = i - 1;
94          }
95          list[i + 1] = temp;                           // Move A[j] to A[i+1].
96      }
97  }
```

Divide the array into a sorted region and an unsorted region. It extracts elements from the unsorted region and inserts them into their correct position within the sorted region. Time complexity：The outer loop executes N-1 times, and the worst case is that the while loop executes i times.

$$\sum_{i=1}^{n-1} i = \frac{n(n-1)}{2} = \Theta(n^2)$$

Space complexity: $\Theta(1)$

- Bubble sort

```
 99  void BubbleSort(char **list,int n){
100      int i, j;
101      char *temp;
102
103      for(i = 0; i <= N-1; i++){                    // Find the smallest item for A[i], 1 ≤ i ≤ n-1
104          for(j = 0; j <N-1-i; j++){                // search among A[i:n]
105              if(strcmp(list[j], list[j+1])>0){     // Swap A[j] and A[j -1], if A[j] < A[j -1]
106                  temp = list[j];
107                  list[j] = list[j+1];
108                  list[j+1] = temp;
109              }
110          }
111      }
112  }
```

Bubble sort pushes the largest element to the rightmost position in each pass. The process continues for N−1 rounds to ensure the entire array is sorted. Time complexity：The outer loop executes N-1 times, and the inner loop executes N-1-i times.

$$\sum_{i=0}^{n-1}(n-1-i)=\frac{n(n-1)}{2}=\Theta(n^2)$$

Space complexity: $\Theta(1)$

- Min-max sort

```
114  void minMaxSort(char **list,int N) {
115      int i;
116      int l=0;                                      // init l and h
117      int h=N-1;
118      char *temp;
119
120      while(l<h){                                   // sort A[l : h]
121          int imin=l;                               // find min and max in A[l : h]
122          int imax=h;
123
124          if (strcmp(list[h],list[l])<0){           // ensure A[h] >= A[l]
125              temp=list[l];
126              list[l]=list[h];
127              list[h]=temp;
128              }
129
130          for(i=l+1;i<h;i++){
131              if (strcmp(list[i], list[imin])<0){
132                  imin=i;
133              }
134
135              else if(strcmp(list[i],list[imax])>0){
136                  imax=i;
137              }
138          }
```

```
139         if(imin>l){                                     // set A[l] = min
140             temp=list[l];
141             list[l]=list[imin];
142             list[imin]=temp;
143         }
144         if(imax<h){                                     // set A[h] = max
145             temp=list[h];
146             list[h]=list[imax];
147             list[imax]=temp;
148         }
149         l=l+1;                                          // reduce sorting range
150         h=h-1;
151     }
152 }
```

Min-Max Sort is an improved version of Selection Sort. Instead of finding just the smallest element in each pass, it simultaneously finds both the smallest and largest elements and places them at opposite ends of the array. It can reduce the number of passes required.

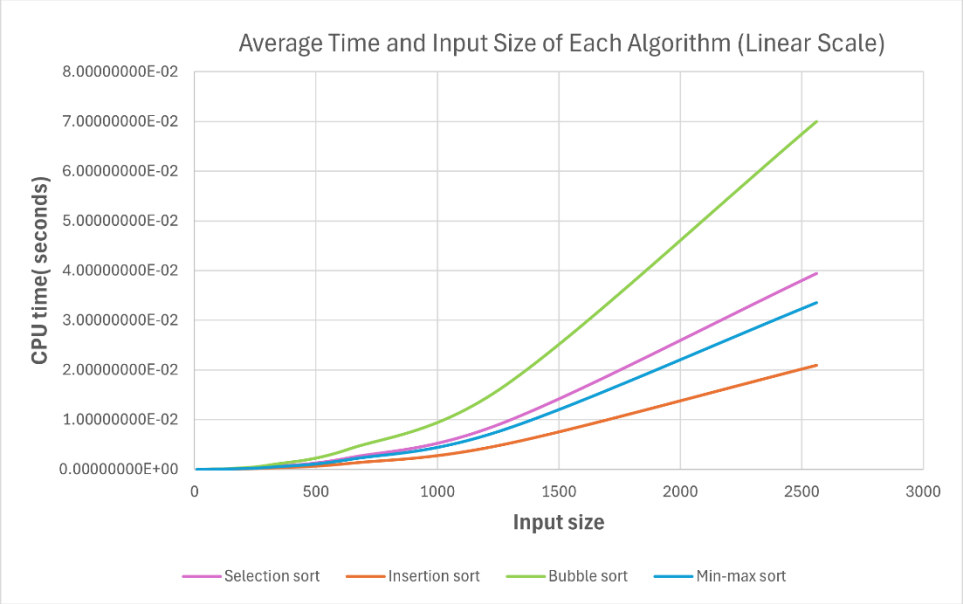
Time complexity：The inner loop executes n-2l times.

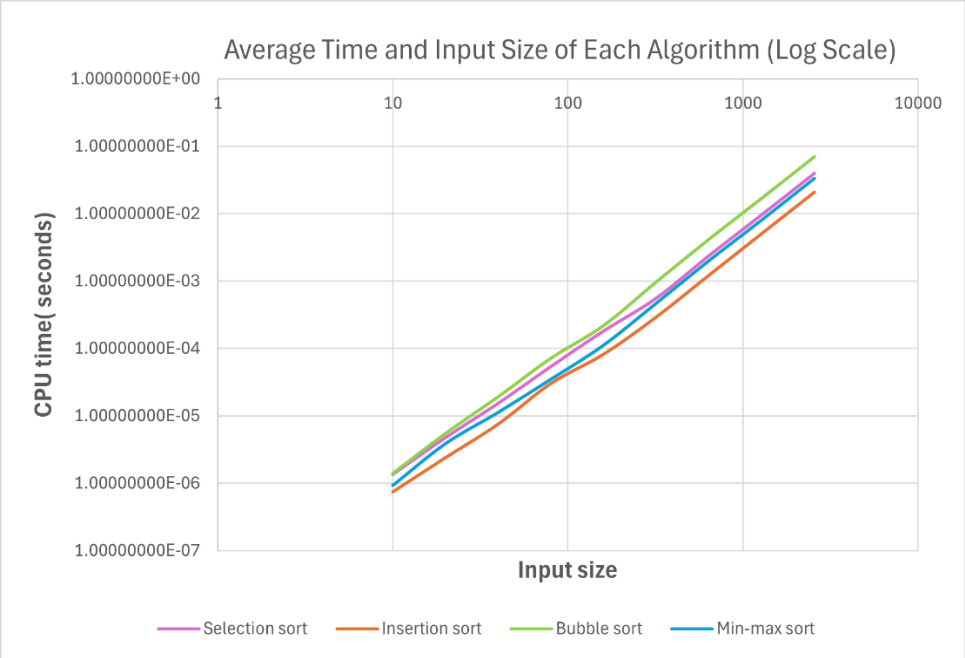
$$\sum_{l=0}^{(n-1)/2} (n-2l) = \Theta(n^2)$$

Space complexity: $\Theta(1)$

- **Results and observations**

Input size	Selection sort	Insertion sort	Bubble sort	Min-max sort
10	1.35183334E-06	7.51972198E-07	1.38998032E-06	9.36031342E-07
20	4.74834442E-06	2.41804123E-06	5.46407700E-06	3.89432907E-06
40	1.52959824E-05	7.60602951E-06	1.91183090E-05	1.13358498E-05
80	5.37123680E-05	3.04241180E-05	7.13520050E-05	3.51500511E-05
160	1.80538177E-04	8.27937126E-05	2.18125820E-04	1.12654209E-04
320	5.54905891E-04	2.94636250E-04	9.69754219E-04	4.72067833E-04
640	2.36687994E-03	1.22751188E-03	4.16615820E-03	2.01757812E-03
1280	9.55609798E-03	5.08632421E-03	1.69290776E-02	8.08513594E-03
2560	3.93826222E-02	2.09051080E-02	6.99726920E-02	3.35067139E-02





Regardless of the input size, bubble sort takes the longest average time among the four algorithms because too many swap operations lead to poor performance. Additionally, min-max sort is slightly better than selection sort in this test because it finds the maximum and minimum values at the same time, so it can reduce the number of searches by half. Moreover, insertion

sort requires only a small number of comparisons and moves. Min-max sort requires more comparisons and exchanges to find the minimum and maximum values separately in each iteration. Therefore, we can know that although these four algorithms have the same time complexity, insertion sort is the best algorithm for this problem.