Embedded System :Assignment#4

Hong Chenhui
`drredthered.github.io`

Zhejiang University — April 16, 2024

Info: Keil is rather...interesting with its own temper compare to normal C compilers. For instance, it won't allow proclamation of any parameters anywhere outside below the main function(i.e. when you tried to declare a i parameter under $BubbleSort$), else it will throw a "C141 syntax error" at you.

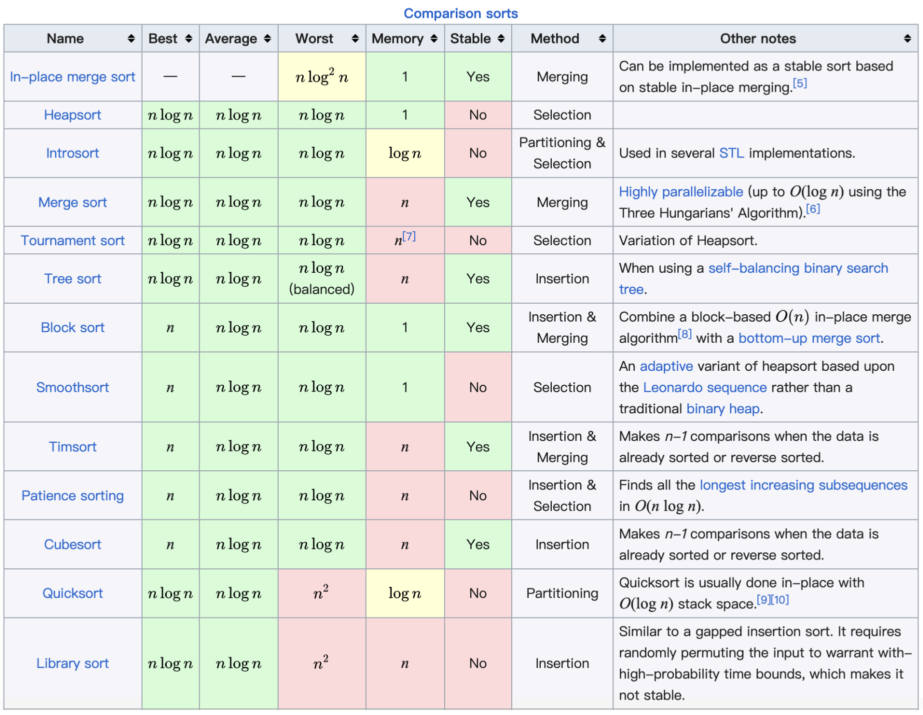
1 Problem 1

- Which sorting algorithm best fits the following scenarios?

  use reg51.h to write a C code that sorts A[15]=(27,5,32,47,38,235,79, 17,187,58,23,35,211,104,9) small to large. The sorted data is stored in RAM.

Answer

Table below demonstrated all kinds of sort with comparison.



Comparison sorts

Name	Best	Average	Worst	Memory	Stable	Method	Other notes
In-place merge sort	—	—	$n \log^2 n$	1	Yes	Merging	Can be implemented as a stable sort based on stable in-place merging.[5]
Heapsort	$n \log n$	$n \log n$	$n \log n$	1	No	Selection	
Introsort	$n \log n$	$n \log n$	$n \log n$	$\log n$	No	Partitioning & Selection	Used in several STL implementations.
Merge sort	$n \log n$	$n \log n$	$n \log n$	n	Yes	Merging	Highly parallelizable (up to $O(\log n)$ using the Three Hungarians' Algorithm).[6]
Tournament sort	$n \log n$	$n \log n$	$n \log n$	n[7]	No	Selection	Variation of Heapsort.
Tree sort	$n \log n$	$n \log n$	$n \log n$ (balanced)	n	Yes	Insertion	When using a self-balancing binary search tree.
Block sort	n	$n \log n$	$n \log n$	1	Yes	Insertion & Merging	Combine a block-based $O(n)$ in-place merge algorithm[8] with a bottom-up merge sort.
Smoothsort	n	$n \log n$	$n \log n$	1	No	Selection	An adaptive variant of heapsort based upon the Leonardo sequence rather than a traditional binary heap.
Timsort	n	$n \log n$	$n \log n$	n	Yes	Insertion & Merging	Makes *n-1* comparisons when the data is already sorted or reverse sorted.
Patience sorting	n	$n \log n$	$n \log n$	n	No	Insertion & Selection	Finds all the longest increasing subsequences in $O(n \log n)$.
Cubesort	n	$n \log n$	$n \log n$	n	Yes	Insertion	Makes *n-1* comparisons when the data is already sorted or reverse sorted.
Quicksort	$n \log n$	$n \log n$	n^2	$\log n$	No	Partitioning	Quicksort is usually done in-place with $O(\log n)$ stack space.[9][10]
Library sort	$n \log n$	$n \log n$	n^2	n	No	Insertion	Similar to a gapped insertion sort. It requires randomly permuting the input to warrant with-high-probability time bounds, which makes it not stable.

Fig.1 Comparison of Sorting Algorithms

For the sake of minimum time and space complexity, Block sorts might be the best choice. Yet because **my stupid engineering brain won't allow me to memorize such a redundant algorithm**, I chose Bubble Sort instead. At worst it might provide $O(n^2)$ time and $O(1)$ space complexity.

2 Implements

language = C

```c
#include <reg51.h>

// Define the array
unsigned int A[15] =
{27, 5, 32, 47, 38, 235, 79,
17, 187, 58, 23, 35, 211, 104, 9};

unsigned short bdata *ptr;

void swap(unsigned int *x, unsigned int *y) {
    unsigned int temp = *x;
    *x = *y;
    *y = temp;
}

void bubbleSort(unsigned int arr[], int n) {
    int i, j;
    for (i = 0; i < n-1; i++) {
        for (j = 0; j < n-i-1; j++) {
            if (arr[j] > arr[j+1]) {
                swap(&arr[j], &arr[j+1]);
            }
        }
    }
}

void main() {
    int i;
/*For some reason this line must be in front of bubble
sort,or else C141 occurs.*/

    bubbleSort(A, 15); // Sort the array

    ptr = 0x40;
/*Because 0x00 will violate with the A array in
RAM, thus messing with the storage.*/
    for (i = 0; i < 15; i++) {
        *ptr = A[i];
        ptr++;
    }

while(1);
}
```

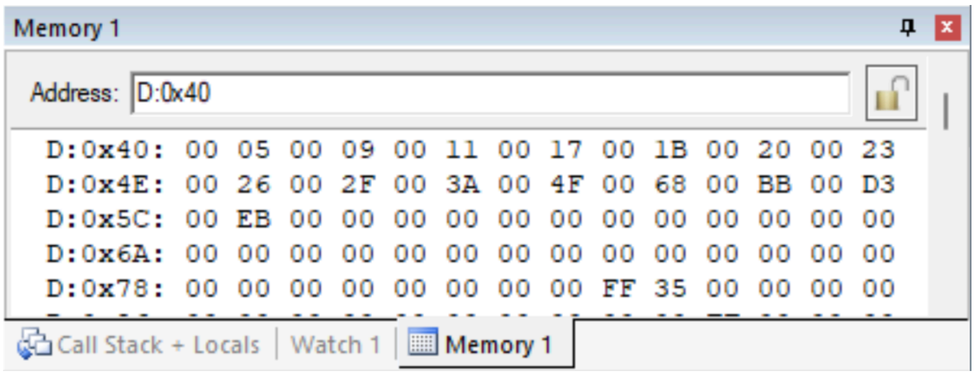

```
Memory 1
Address: D:0x40
D:0x40: 00 05 00 09 00 11 00 17 00 1B 00 20 00 23
D:0x4E: 00 26 00 2F 00 3A 00 4F 00 68 00 BB 00 D3
D:0x5C: 00 EB 00 00 00 00 00 00 00 00 00 00 00 00
D:0x6A: 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D:0x78: 00 00 00 00 00 00 00 00 FF 35 00 00 00 00
```

Fig.2 Sorted output in RAM

3 Problem 2

Describe how convenient it is to write C51 program with C instead of Assembly language.

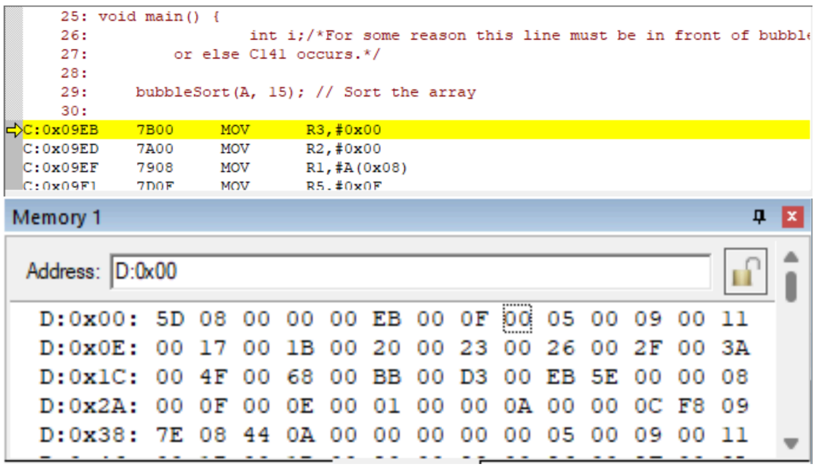
Answer

1. Writing code in C is generally faster than writing in assembly language. C compilers handle many low-level details such as register allocation, instruction optimization, and memory management automatically, saving one's time and effort.
2. C supports structured programming constructs like functions, loops, conditionals, and data structures (arrays, structs), which enable one to organize code in a more modular and logical manner. This improves code clarity and makes it easier to manage larger projects.
3. C is a higher-level language compared to assembly, which means one can express algorithms and logic more concisely and clearly. This higher level of abstraction makes it easier to understand, debug, and maintain code.

4 Problem 3

- On what address did the program put the original array A?

Answer


```
    25: void main() {
    26:                 int i;/*For some reason this line must be in front of bubble
    27:         or else C141 occurs.*/
    28:
    29:     bubbleSort(A, 15); // Sort the array
    30:
C:0x09EB    7B00    MOV     R3,#0x00
C:0x09ED    7A00    MOV     R2,#0x00
C:0x09EF    7908    MOV     R1,#A(0x08)
C:0x09F1    7D0F    MOV     R5,#0x0F
```
```
Memory 1
Address: D:0x00
D:0x00: 5D 08 00 00 00 EB 00 0F 00 05 00 09 00 11
D:0x0E: 00 17 00 1B 00 20 00 23 00 26 00 2F 00 3A
D:0x1C: 00 4F 00 68 00 BB 00 D3 00 EB 5E 00 00 08
D:0x2A: 00 0F 00 0E 00 01 00 00 0A 00 00 0C F8 09
D:0x38: 7E 08 44 0A 00 00 00 00 00 05 00 09 00 11
```

Apparently 0x08.