Trần Tinh Chí
1351063
ECE341 – Lab02

Lab02-Readme

Because I have described in the assembly code (and write down everything I coded in MIPS is too painful, so I think I just show you the code I translated from C++ to MIPS, and you can compare it to my MIPS code and understand it via comments in .asm file and in .cpp file), so I just show you the C++ version of these Search and Sort algorithms. **Note that all of them were written down by me in both C++ and MIPS languages**.

Linear Search:

Binary Search:

```
      //This is the implement function, not the interface function. The interface
function is simple so I do not bring it to this document.
    int ImplementBinarySearch (int * a, int n, int x, int left, int right) {
        if (left > right) //left > right likes n > (n+1)
            return -1;

        int middle = (left + right) / 2;

        if (a[middle] == x) //If the x value is found
            return middle;
        else if (a[middle] < x) //Search in right-half
            return ImplementBinarySearch (a, n, x, middle + 1, right);
        else  //Search in left-half
            return ImplementBinarySearch (a, n, x, left, middle - 1);
    }
```

Bubble Sort:

```
      //Ascending order version
    void BubbleSort (int * a, int n) {
        for (int i = 1 ; i < n ; i++) {
            for (int j = n - 1 ; j >= i ; j--) {
                if (a[j] < a[j-1])
                    Swap (a[j], a[j-1]) ; //Swap if decreasing
            }
        }
    }
```

Insertion Sort:

```
      //Descending order version
    void InsertionSort (int * a, int n) {
        for (int i = 1, j ; i < n ; i++) {
            int temp = a[i];
            for (j = i - 1 ; j >= 0 && temp > a[j] ; j--)
                a[j+1] = a[j]; //Shifting forward

            a[j+1] = temp;
        }
    }
```

Divide Positive Negative:

```
// Used to divide the array A into 2 arrays nega and notnega, where nega keeps
the negative numbers, and notnega keeps the non-negative numbers.
void DividePositiveNegative (int * a, int n, int * &nega, int & n_nega, int *
&not_nega, int & n_not_nega){

    n_nega = n_not_nega = 0;
    int j = 0, k = 0;
    for (int i = 0; i < n ; ++i) {
        if (a[i] < 0) {
            // If negative => keep in negative array
            nega[j] = a[i];
            ++j;
        }
        else {
            // Otherwise, keep in non-negative array
            not_nega[k] = a[i];
            ++k;
        }
    }
    n_nega = j;
    n_not_nega = k;
}
```

Merge Sort:

```
    //Strategy for Question 3: I divide array 'arr' into to 'nega' and
'not_nega', then sort 'nega' with descending merge sort and 'not_nega' with
ascending merge sort. After that, print out 'nega' next to 'not_nega'.

    //Ascending and Descending version
void MergeSort(int * a, int n, bool asc) {
    if (n <= 1) return;

    int * b = &a[n / 2];
  //Divide the first half
    MergeSort(a, n / 2, asc);
    int * temp1 = new int[n / 2];
    for (int i = 0; i < n / 2; i++)
        temp1[i] = a[i];

  //In MIPS code version of mine for Merge sort, I have used stack to backup
the content of array temp1, but haven't done the same thing for array temp2,
why I have to do that? Because the content of array temp1 is not guaranteed
after the 2nd recursive call. It maybe replaced by the child call. So, I push
the content into stack then recover them immediately after 2nd recursive call.
But with temp2, I don't have to backup its content, because after the 2nd
recursive call, we have no recursive call, it means that after some done all
linear code below, the current procedure will finish and backtrack to its
parent procedure. In the parent one, this child maybe the 1st recursion ->
it's fine because we haven't need temp2 yet, or this child maybe the 2nd
recursion -> temp2 will be calculated now.

  //Divide the second half
    MergeSort(b, n - n / 2, asc); //Different from n/2
    int * temp2 = new int[n - n / 2];
    for (int i = 0; i < n - n / 2; i++)
        temp2[i] = b[i];

  //All the code below just merges temp1 and temp2 into 1 array with ascending
order.
    int i, j, k;
    for (i = 0, j = 0, k = 0; i < n; i++){
```

```
            if (j < n / 2 && k < n - n / 2) {
                if (asc) {
                    //Ascending version
                  if (temp1[j] < temp2[k]) {
                      a[i] = temp1[j];
                      j++;
                  }
                  else {
                      a[i] = temp2[k];
                      k++;
                  }
                }
                else {
                    //Descending version
                    if (temp1[j] > temp2[k]) {
                       a[i] = temp1[j];
                       j++;
                    }
                    else {
                       a[i] = temp2[k];
                       k++;
                    }
                }

            }
            else break;
        }

        if (j >= n / 2){
            if (k > j)
                i = k;

            for (; i < n; i++, k++)
                a[i] = temp2[k];
        }
        else if (k >= n - n / 2){
            if (j > k)
                i = j;

            for (; i < n; i++, j++)
                a[i] = temp1[j];
        }

        delete[] temp2, temp1;
    }
```

Quick Sort:

```
      //Ascending version
     void ImplementQuickSort(int * a, int start, int end){
        if (start >= end) //start and end converge
            return;
        int pivot = a[(start + end) / 2];
        int i = start, j = end;

        do {
            while (a[i] < pivot) ++i;
            while (a[j] > pivot) --j;

            if (i <= j) {
                if (i < j)
                    Swap(a[i], a[j]);
                i++, j--;
            }
```

```
        } while (i <= j);
        ImplementQuickSort(a, start, j);
        ImplementQuickSort(a, i, end);

    }

    void QuickSort(int * a, int n){
        ImplementQuickSort(a, 0, n - 1);
    }
```

Joint Array Interpose:

```
      //Used to join 2 array with interpose order (xen kẽ nhau)
    int * JointArrayInterpose (int * a, int na, int * b, int nb){
        int i = 0;
        int j = 0;

        int * c = new int [na + nb];

        for (int k = 0 ; k < na + nb;) {
            if (i < na) {
                c[k] = a[i];
                ++i;
                k++;
            }
            if (k < na + nb && j < nb) {
                c[k] = b[j];
                ++j;
                k++;
            }
        }

        return c;
    }
```