Spencer York
Dr. Norstrom
Data Structures
14 April 2019

Sorting

Q1: Assume you were given a task at your next programming job to sort a collection of objects. What are the first three questions you should answer before starting to write the code?

- How much data needs to be sorted?
- Is the data random or already somewhat sorted?
- Can the data fit onto the main memory or will it be sorted on slower, external memory?

Q2: Having answered those three questions, is that enough to make a definite decision as to which sorting algorithm you will use?

- After answering those three questions, it might not provide you with a definite answer, but it will lead you towards which algorithm to choose

Q3: Describe a hybrid sorting algorithm that is good at both very large and small collection sizes. Give the pidgin-or pseudo-code for this hybrid algorithm.

When I first looked at this question, my first thought was to somehow combine insertion sort and merge sort. After doing some research, I learned that Timsort is a very stable hybrid sort that does just that. It is very good at sorting both small and large amounts of data. The code for this sorting algorithm is below:

- We consider size of run as 32.
- We one by one sort pieces of size equal to run
- After sorting individual pieces, we merge them one by one. We double the size of merged subarrays after every iteration.

```
const int RUN = 32;
// this function sorts array from left index to
// to right index which is of size atmost RUN

void insertionSort(int arr[], int left, int right)
{
    for (int i = left + 1; i <= right; i++)
    {
        int temp = arr[i];
        int j = i - 1;
        while (arr[j] > temp && j >= left)
        {
            arr[j+1] = arr[j];
            j--;
        }
        arr[j+1] = temp;
    }
}

// merge function merges the sorted runs
void merge(int arr[], int l, int m, int r)
{
    // original array is broken in two parts
    // left and right array
    int len1 = m - l + 1, len2 = r - m;
    int left[len1], right[len2];
    for (int i = 0; i < len1; i++)
        left[i] = arr[l + i];
    for (int i = 0; i < len2; i++)
        right[i] = arr[m + 1 + i];

    int i = 0;
    int j = 0;
    int k = l;

    // after comparing, we merge those two array
    // in larger sub array
    while (i < len1 && j < len2)
    {
        if (left[i] <= right[j])
        {
            arr[k] = left[i];
            i++;
        }
        else
        {
            arr[k] = right[j];
            j++;
        }
        k++;
    }

    // copy remaining elements of left, if any
    while (i < len1)
    {
        arr[k] = left[i];
        k++;
        i++;
    }

    // copy remaining element of right, if any
    while (j < len2)
    {
        arr[k] = right[j];
        k++;
        j++;
    }
}

// iterative Timsort function to sort the
// array[0...n-1] (similar to merge sort)
void timSort(int arr[], int n)
{
    // Sort individual subarrays of size RUN
    for (int i = 0; i < n; i+=RUN)
        insertionSort(arr, i, min((i+31), (n-1)));

    // start merging from size RUN (or 32). It will merge
    // to form size 64, then 128, 256 and so on ....
    for (int size = RUN; size < n; size = 2*size)
    {
        // pick starting point of left sub array. We
        // are going to merge arr[left..left+size-1]
        // and arr[left+size, left+2*size-1]
        // After every merge, we increase left by 2*size
        for (int left = 0; left < n; left += 2*size)
        {
            // find ending point of left sub array
            // mid+1 is starting point of right sub array
            int mid = left + size - 1;
            int right = min((left + 2*size - 1), (n-1));
            // merge sub array arr[left.....mid] &
            // arr[mid+1....right]
            merge(arr, left, mid, right);
        }
    }
}
```