Data Structures Term Project

Iterative Merge Sort

Yiğit Berk Sarıboğa

090190354

Arif Çakır

090190355

Ahmet Salih Coşkun

090190356

Deniz Kaan Şahiner

090190357

Nizameddin Ömeroğlu

090190358

Department of Mathematical Engineering, Istanbul Technical University

MAT 226E: Data Structures, CRN: 21372

Assistant Professor İzzet Göksel

May 7, 2022

Objective of the Project

In this project, the main objective is to analyze the iterative merge sort algorithm and then explain it with the help of materials such as a C++ code, a report, a YouTube video, and finally an in-class presentation. For that purpose, this report includes four sections to analyze and present the topic to the reader. In section 1, the iterative merge sort algorithm and its components are evaluated. After that, section 2 includes key parts of the C++ code with the comments of the team's coder. Properties of iterative merge sort are discussed in section 3, while section 4 only contains a YouTube link for the video.

Distribution of the Roles

- Captain: Yiğit Berk Sarıboğa
- Coder: Deniz Kaan Şahiner
- Moviemaker: Ahmet Salih Coşkun
- Presenter: Nizameddin Ömeroğlu
- Reporter: Arif Çakır

Algorithm

Iterative merge sort, also known as bottom-up merge sort, is an implementation of the merge sort algorithm that avoids recursion by using iteration. First, it is necessary to understand the merge sort algorithm (MSA) to understand the iterative merge sort algorithm. According to Sanderson (2020), MSA is a divide and conquer algorithm because it takes relatively larger data and divides it into tinier and workable data pieces to solve. Recursive MSA differs from the iterative merge sort algorithm because it uses recursion instead of iteration to reach and sort smaller data pieces. As Ryder (n.d.) highlighted, John von Neumann is the inventor of MSA in 1945.

Iterative merge sort algorithm (IMSA) goes from bottom to top, which means it actually starts to sort from 1 element arrays to the whole array, but due to 1 element arrays being already sorted, it starts to sort from 2 element arrays. After that, the results of these sorts are merged into 4 element arrays and it continues for n=2,4,8,16… values.

According to Tiwari (n.d.), for an array Arr[] with size n, IMSA can be summed as:

1. Initialize sub_size and assign it to 1 (for the first time). If sub_size is less than n/2, multiply it by 2 and continue.

2. Initialize Left and assign it to 0 (for the first time). Add 2*sub_size to Left as long as it is less than n. Initialize Mid value as minimum of (Left+sub_size-1, n-1) and Right as minimum of (Left + 2*sub_size -1, n-1).

3. Initialize A and copy sub_array [Left, Mid-1] to A, then initialize B and copy sub array [Mid, Right] to B.
    3.1. Compare first elements of A and B, then remove the first element of that array and add it to array C.
    3.2. Repeat 3.1 until A or B is empty.
    3.3. Copy the non-empty list to C.

4. Copy C to Arr[] from Left to Right. Return to 1.

Key Parts of the Code and Comments

```
void merge(int arr[], int l, int m, int r){
   int i, j, k;
   int n1 = m - l + 1;
   int n2 =  r - m;

   // Creates temp arrays
   int L[n1], R[n2];

   // Copies the data to temp arrays as L[] and R[]
```

```
for (i = 0; i < n1; i++)
    L[i] = arr[l + i];
for (j = 0; j < n2; j++)
    R[j] = arr[m + 1+ j];

// Merges the temp arrays back into arr[l..r] as ascending order
i = 0;
j = 0;
k = l;
while (i < n1 && j < n2){
    if (L[i] <= R[j]){
        arr[k] = L[i];
        i++;}
    else{
        arr[k] = R[j];
        j++;}
    k++;}

// If there are any remaining elements of array l, copy these elements
while (i < n1){
    arr[k] = L[i];
    i++;
    k++;}

// If there are any remaining elements of array r, copy these elements
while (j < n2){
    arr[k] = R[j];
    j++;
    k++;}}
```

Properties of Iterative Merge Sort

- Time Complexity: The time complexity of the iterative merge sort is O(n*log(n). For calculating the time complexity of IMSA for an array with size n, according to Aatrey, firstly the number of rows comes into consideration. It starts from 1 element arrays and then multiply them by 2 until size is equal to n. Mathematically speaking:

  1*(2*2*2*…*2) = n
  2^ (number of rows) = n
  number of rows = $\log_2 n$

  Which approximates to O(log(n)).

  After that, it is needed to calculate the time complexity of merging two arrays. The merge algorithm compares the first elements of the arrays and removes smaller one from the array to continue (Aatrey, n.d.). Due to that, the number of operations to empty both arrays is equal to the sum of sizes of the both arrays. Aatrey continues as eventually sum of all arrays will be equal to the size of the starting array. Because of that, the time complexity of this operation is O(n). If first part is repeated for all sub arrays, then time complexity of whole algorithm is O(n)*O(log(n)) = O(n*log(n)).

- Auxiliary Space: Iterative merge sort uses auxiliary space only for holding the results of merged arrays and because of that, it is equal to the sum of the sizes of merged arrays. At worst, the sum is equal to the final array, thus auxiliary space of iterative merge sort is O(n).

- In-place: As Aatrey highlights, iterative merge sort is not an in-place algorithm because auxiliary space is used for holding the results of merged arrays.

- Stable: Yes, iterative merge sort algorithm is stable because order of same elements stays the same as start.

- Online: No, iterative merge sort algorithm is an offline sorting algorithm, elements checked from the first element of array.

Video

YouTube link for the video: https://youtu.be/-uMcbPaQBe0

Conclusion

In general, iterative merge sort is a sorting algorithm that has a divide and conquer approach to the data. Due to that, it helps dividing greater data into smaller data pieces to work with. Like all things in computer science, it has advantages like its time complexity being O(n*log(n)) and disadvantages like not being an in-place sorting algorithm. These kinds of sorting algorithms are tools of a programmer's toolbox and can be selected for best use by them.

References

Sanderson, T. (2020, April 7). *Merge Sort Algorithm 101.* The Startup. https://medium.com/swlh/merge-sort-algorithm-101-c4fdea276289

Ryder, A. (n.d.). *Merge Sort*. OpenGenus IQ. https://iq.opengenus.org/merge-sort/

Tiwari, A. (n.d.). *Iterative Merge Sort.* Interview Kickstart. https://www.interviewkickstart.com/learn/iterative-merge-sort

GeeksforGeeks. (2022, May 2). *Iterative Merge Sort.* https://www.geeksforgeeks.org/iterative-merge-sort/

Aatrey, T. S. *Merge Sort Algorithm.* Interview Kickstart. https://www.interviewkickstart.com/learn/merge-sort

Appendix

```
/* Iterative C program for merge sort */

#include <bits/stdc++.h>

using namespace std;
```

Iterative Merge Sort

```
// Function to merge the two subarrays like arr[l..m] and arr[m+1..r] of array arr[]

void merge(int arr[], int l, int m, int r);



int min(int x, int y) { return (x<y)? x :y; } // finding minimum of two integers. (if x is smaller than y, use x else use y)



// Iterative Merge Sort Function

void mergeSort(int arr[], int n){

  int curr_size;  //specifies the current size of subarrays to be merged, varies between 1 and n/2.


  int left_start; // to pick starting value of left subarray to be merged.


// Merges subarrays from bottom to top. Subarrays of size 1 merges and creates sorted subarrays of size 2 and this process continues.

  for (curr_size=1; curr_size<=n-1; curr_size = 2*curr_size){

     //picks starting points for left subarrays based on current size

     for (left_start=0; left_start<n-1; left_start += 2*curr_size){


        int mid = min(left_start + curr_size - 1, n-1);  // mid is ending point of left subarray and mid+1 equals to starting point of right subarray.


        int right_end = min(left_start + 2*curr_size - 1, n-1); //right_end is last point of right subarray.


        // merges subarrays like -> arr[left_start...mid] & arr[mid+1...right_end]

        merge(arr, left_start, mid, right_end);

     }
  }
}

// Function to merge the two subarrays like arr[l..m] and arr[m+1..r] of array arr[]
void merge(int arr[], int l, int m, int r)
{
    int i, j, k;
    int n1 = m - l + 1;
    int n2 =  r - m;

    // Creates temp arrays
    int L[n1], R[n2];

    // Copies the data to temp arrays as L[] and R[]
    for (i = 0; i < n1; i++)
        L[i] = arr[l + i];
    for (j = 0; j < n2; j++)
        R[j] = arr[m + 1+ j];

    // Merges the temp arrays back into arr[l..r] as ascending order
    i = 0;
    j = 0;
    k = l;
    while (i < n1 && j < n2)
    {
        if (L[i] <= R[j])
        {
            arr[k] = L[i];
            i++;
        }
        else
        {
            arr[k] = R[j];
            j++;
        }
        k++;
    }

    // If there are any remaining elements of array l, copy these elements
    while (i < n1)
    {
        arr[k] = L[i];
        i++;
        k++;
    }

    // If there are any remaining elements of array r, copy these elements
    while (j < n2)
    {
        arr[k] = R[j];
        j++;
        k++;
    }
}

// function that prints the array
```

```
void printArray(int A[], int size)
{
   int i;
   for (i=0; i < size; i++)
      cout <<" "<< A[i];
   cout <<"\n";
}

// testing phase
int main()
{
   int arr[] = {72,25,34,13,-3,6,61};
   int n = sizeof(arr)/sizeof(arr[0]);

   cout <<"Given array is \n ";
   printArray(arr, n);

   mergeSort(arr, n);

   cout <<"\nSorted array by iterative merge sort algorithm is \n ";
   printArray(arr, n);
   return 0;
}
// This code is contributed shivanisinghss2110
// https://www.geeksforgeeks.org/iterative-merge-sort
```