

IIT Madras

ONLINE DEGREE

Programming, Data Structures and Algorithms Using Python
Professor. Madhavan Mukund
Merge Sort

(Refer Slide Time: 0:09)

Beating the $O(n^2)$ barrier

- Both selection sort and insertion sort take time $O(n^2)$
- This is infeasible for $n > 10000$
- How can we bring the complexity below $O(n^2)$?

Madhavan Mukund Merge Sort

So, we have seen two intuitive algorithms, insertion sort and selection sort. And both of them happen to be n squared. And we know that n squared is not good because if we go beyond say something like n to the power 4 already, you will start getting algorithms which take too long to be feasible in practice.

So, we would like something which is better than n squared. So, how do we do this? So, we need some other strategy and so here is a strategy which is perhaps less intuitive than the one that we use for selection and Insertion Sort. So, in this strategy, we divide the list into two parts into two equal parts, in fact, two halves, we separately sort the left half and the right half.

So, if we are still thinking in terms of that TA problem that we had, so this instructor gives the pile of papers to the TA, supposing the instructor has two TA, then the instructor can give half the papers to one TA, and the other half to the other TA and notice that these two halves can be sorted without reference to each other.

So, each of them sorts their own half within that half. But now the two halves have to be reconciled, because some papers in the bigger half in the first half might have lower marks and some in the second half, and so on. So, then we have to take the two halves and combine them to

get a fully sorted list. So, there is some work to be done to combine the individual TA answers or the TA sorted bundles into one full sorted bundle. So, this is the strategy that we are going to look at now.

(Refer Slide Time: 1:34)

Combining two sorted lists

■ Combine two sorted lists **A** and **B** into a single sorted list **C**

- Compare first elements of **A** and **B**
- Move the smaller of the two to **C**
- Repeat till you exhaust **A** and **B**

Top Bottom

32 74 89
21 55 64

Madhavan Mukund Merge Sort

Combining two sorted lists

■ Combine two sorted lists **A** and **B** into a single sorted list **C**

- Compare first elements of **A** and **B**
- Move the smaller of the two to **C**
- Repeat till you exhaust **A** and **B**

32 74 89
21 55 64
21

Madhavan Mukund Merge Sort

Combining two sorted lists



- Combine two sorted lists A and B into a single sorted list C
- Compare first elements of A and B
- Move the smaller of the two to C
- Repeat till you exhaust A and B

32 74 89
21 55 64
21 32 55



Combining two sorted lists



- Combine two sorted lists A and B into a single sorted list C
- Compare first elements of A and B
- Move the smaller of the two to C
- Repeat till you exhaust A and B
- Merging A and B

32 74 89
21 55 64
21 32 55 64 74 89



So, in order to do this, let us focus on the second part, which is the two TA of come back and they have brought me two sorted bundles of answer books, how do I combine them into a single sorted bundle. So, I take two sorted lists A and B, and I want to combine it into a single sorted list, C.

So, this is something you can imagine, again, physically, you put them down, so each of them is sorted, with the highest mark on top. So, which is the highest mark overall? Well, it is either the highest mark in the second pile or the highest mark in the first pile. So, I look at the top two, and I decide, oh, this is higher than this. So, this must be the highest overall and I put it aside.

Now I compare the highest here and the second paper that is visible here, again, the higher of the two must be the highest overall and what is here, so I moved that there. So, now I will kind of in some sense, in this physical process, I am reversing the order, but the highest paper comes down, then the second highest paper comes down and so on. So, by gradually, you know, running through these two piles systematically, I will be able to take them and run them down to get the pile in a single sorted order.

So, I compare the first element move the smaller of the two or the bigger of the two to C and repeat until you exhausted. So, let us look at this example. Supposing this happened, a trivial thing. So, the first pile has only three papers, 32, 74, 89 and let us say that 32 and 21 are the papers which I can see. So, they are the top of the pile.

So, in my earlier thing, so this is slightly. So, this is now my top. So, I start from the left hand side. So, I compare 21 and 32 and I want the new list to also be in ascending order. So, I compared 21 and 32 and the smaller of the two is the smallest overall and move the 21 out. Now I am focusing on the 55 compared to the 32.

So, these are the two that are visible to me, if they were physically sitting there, I moved the smallest one out. And this is what is visibly there. So, I now find the 32 smaller and remove it. Now I am looking at these two, because the other two are gone. So, now, 55 is smaller. Now I am looking at these two.

Now 64 is smaller and finally I moved 74 and 89 because there is nothing to compare them with anymore. So, I only have to move from the first pile. So, this is how I would combine two sorted lists into a single sorted list. So, this is what is called merging. So, merging is the operation that I need after I have got the things sorted by two different TA.

(Refer Slide Time: 4:04)

Merge sort



- Let n be the length of L

- Sort $A[:n//2]$

- Sort $A[n//2:]$

- Merge the sorted halves into B

- How do we sort $A[:n//2]$ and $A[n//2:]$?

- Recursively, same strategy!

43 32 22 78 63 57 91 13

43 32 22 78 63 57 91 13

43 32 22 78 63 57 91 13

43 32 22 78 63 57 91 13

Navigation icons

Madhavan Mukund

Merge Sort

Merge sort



- Let n be the length of L

- Sort $A[:n//2]$

- Sort $A[n//2:]$

- Merge the sorted halves into B

- How do we sort $A[:n//2]$ and $A[n//2:]$?

- Recursively, same strategy!

43 32 22 78 63 57 91 13

43 32 22 78 63 57 91 13

32 43 22 78 63 57 91 13

43 32 22 78 63 57 91 13

Navigation icons

Madhavan Mukund

Merge Sort

सिद्धिर्भवति कर्मजा

Merge sort



- Let n be the length of L

- Sort $A[:n//2]$

- Sort $A[n//2:]$

- Merge the sorted halves into B

- How do we sort $A[:n//2]$ and $A[n//2:]$?

- Recursively, same strategy!

43	32	22	78	63	57	91	13
----	----	----	----	----	----	----	----

43	32	22	78
----	----	----	----

63	57	91	13
----	----	----	----

32	43
----	----

22	78
----	----

57	63
----	----

13	91
----	----

43	32	22	78	63	57	91	13
----	----	----	----	----	----	----	----

Navigation icons: back, forward, search, etc.

Madhavan Mukund

Merge Sort

So, now, I have to get the thing to sort by these two different TA. So, how do I do it, this is the Merge Sort part. So, I let n be the length. So I will first start the beginning, and then I will sort the end. So, if I take the length and divide by 2, then I will get two segments of roughly equal length. So, I will sort the first half and sort the second half.

And then I will merge using the procedure, which I just described how to take to merge links and bring, give me back a fully merged list. So, I am merging these two sorted half into B . And how do we sort these two half? Well, we use the same procedure that is we will take each of them break them up into two parts. Sort the first half or the second half merge. How will we do that?

Again, we will break it up into two parts. And when do we stop when we stop? When we come to a list which has only one element when it has only one element, then there is no more work to do. So, let us look at an example.

So, supposing we have eight elements then I do not know how to sort these eight elements. So I divided into the first half and the second half, and I asked my two TA to sort them, so they do not know how to sort them. So, they find two friends, and they pass them each, the two halves of the first half gets split into two lengths of two, the second half gets split into two lengths of two. So now I have 4 parts with to each.

Now these two friends do not know how to do this either. So they each call two more friends, and give them one paper each. So now basically, this has come like this. So this is how they have

split. So now the last guy in this list, the last set of TAs who have been roped into this job have an easy job, because they have just got one paper.

So they say, Oh, this is easy, I can just give it back to you. So Mr. 43 says I am done. Mr. 32 says I am done, and so on. So, all these last eight TA report that they have sorted their work. So now the second level TA have to merge them. So I have to take these two and merge them. So if I merge them, then 32 is smaller than 43.

So I will replace the list at this level. By the sorted version, I will do the same thing with 22 and 78. Here, nothing changes. The next one 157 comes before 63. The next one 113 comes before 91. So now I am finished with this level, and I can throw away the singleton thing. So all those TA and are dismissed.

(Refer Slide Time: 6:25)

Merge sort

■ Let n be the length of L

■ Sort $A[:n//2]$

■ Sort $A[n//2:]$

■ Merge the sorted halves into B

■ How do we sort $A[:n//2]$ and $A[n//2:]$?

- Recursively, same strategy!

43 32 22 78 63 57 91 13

43 32 22 78 63 57 91 13

32 43 22 78 57 63 13 91

Madhavan Mukund Merge Sort

Merge sort



- Let n be the length of L

- Sort $A[:n//2]$

- Sort $A[n//2:]$

- Merge the sorted halves into B

- How do we sort $A[:n//2]$ and $A[n//2:]$?

- Recursively, same strategy!

43 32 22 78 63 57 91 13

22 32 43 78 63 57 91 13

32 43 22 78 57 63 13 91

Navigation icons

Madhavan Mukund

Merge Sort



Merge sort



- Let n be the length of L

- Sort $A[:n//2]$

- Sort $A[n//2:]$

- Merge the sorted halves into B

- How do we sort $A[:n//2]$ and $A[n//2:]$?

- Recursively, same strategy!

43 32 22 78 63 57 91 13

22 32 43 78 13 57 63 91

32 43 22 78 57 63 13 91

Navigation icons

Madhavan Mukund

Merge Sort



सिद्धिर्भवति कर्मजा

Merge sort



- Let n be the length of L

- Sort $A[:n//2]$

- Sort $A[n//2:]$

- Merge the sorted halves into B

- How do we sort $A[:n//2]$ and $A[n//2:]$?

- Recursively, same strategy!

43 32 22 78 63 57 91 13

22 32 43 78 13 57 63 91

32 43 22 78 57 63 13 91

Madhavan Mukund Merge Sort

So, now I have the sorted list of length two. So, I combined the sorted list of length two into two sorted list or list of length four. So, from the left hand side, I now get the list 22, 32, 43, 78, which is the sorted version of this. And from the right hand side, I get 13, 57, 63, 91, which is a sorted version of this, how do I get a sorted version of this?

I apply that merge function, I look at the smaller of the two, I pull out the 13, then I pull out 57, then I pull out the 63. And I pull out the 91. Now again, I have got these two sorted sequences of length 4, so I can throw away this smaller sequences that I had used in between.

(Refer Slide Time: 7:01)

Merge sort



- Let n be the length of L

- Sort $A[:n//2]$

- Sort $A[n//2:]$

- Merge the sorted halves into B

- How do we sort $A[:n//2]$ and $A[n//2:]$?

- Recursively, same strategy!

43 32 22 78 63 57 91 13

22 32 43 78 13 57 63 91

13 22 32 43 57 63 78 91

Madhavan Mukund Merge Sort

Merge sort



- Let n be the length of L

- Sort $A[:n//2]$

13	22	32	43	57	63	78	91
----	----	----	----	----	----	----	----

- Sort $A[n//2:]$

- Merge the sorted halves into B

22	32	43	78
----	----	----	----

13	57	63	91
----	----	----	----

- How do we sort $A[:n//2]$ and $A[n//2:]$?

- Recursively, same strategy!

Navigation icons

Madhavan Mukund

Merge Sort

Merge sort



- Let n be the length of L

- Sort $A[:n//2]$

13	22	32	43	57	63	78	91
----	----	----	----	----	----	----	----

- Sort $A[n//2:]$

- Merge the sorted halves into B

- How do we sort $A[:n//2]$ and $A[n//2:]$?

- Recursively, same strategy!

Navigation icons


Madhavan Mukund

Merge Sort

And now I do a similar merge here. So I say, I want 13 first, then I want 22, then I want 32, then I want 43, then I want 57. Then I want 63 then I want 78 and then I want 91. So, this is my merge. So, if I apply this merge, I get the sorted sequence on top, and I am done. So, then I can throw away this and this is my final answer. So this is how merge sort works.

(Refer Slide Time: 7:33)


Merge sort



- Let n be the length of L
- Sort $A[:n//2]$
- Sort $A[n//2:]$
- Merge the sorted halves into B
- How do we sort $A[:n//2]$ and $A[n//2:]$?
 - Recursively, same strategy!

Divide and Conquer

- Break up the problem into disjoint parts
- Solve each part separately
- Combine the solutions efficiently




Madhavan Mukund Merge Sort

So, Merge Sort is an example of something called divide and conquer, you break up the problem into disjoint parts solve each part separately, remember, the two TA could go off into their respective places and work without having to talk to each other. And then when the solutions are done, you have to combine. That is a general step.

So, there are two parts. One is breaking it up. This case breaking it up was easier; I just take the first half in the second half. And then there is a combining part, I take the solution given to me by the first TA, solution given the second TA and I have to merge them.


(Refer Slide Time: 8:06)

Merging sorted lists



- Combine two sorted lists A and B into C
 - If A is empty, copy B into C
 - If B is empty, copy A into C
 - Otherwise, compare first elements of A and B
 - Move the smaller of the two to C
 - Repeat till all elements of A and B have been moved

```
def merge(A,B):  
    m,n = (len(A),len(B))  
    (C,i,j,k) = ([],0,0,0)  
    while k < m+n:  
        if i == m:  
            C.extend(B[j:])  
            k = k + (n-j)  
        elif j == n:  
            C.extend(A[i:])  
            k = k + (n-1)  
        elif A[i] < B[j]:  
            C.append(A[i])  
            (i,k) = (i+1,k+1)  
        else:  
            C.append(B[j])  
            (j,k) = (j+1,k+1)  
    return(C)
```



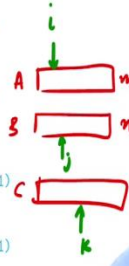
Madhavan Mukund Merge Sort

Merging sorted lists



- Combine two sorted lists *A* and *B* into *C*
 - If *A* is empty, copy *B* into *C*
 - If *B* is empty, copy *A* into *C*
 - Otherwise, compare first elements of *A* and *B*
 - Move the smaller of the two to *C*
 - Repeat till all elements of *A* and *B* have been moved

```
def merge(A,B):
    (m,n) = (len(A),len(B))
    (i,j,k) = (0,0,0)
    while k < m+n:
        if i == m:
            C.extend(B[j:])
            k = k + (n-j)
        elif j == n:
            C.extend(A[i:])
            k = k + (m-i)
        elif A[i] < B[j]:
            C.append(A[i])
            (i,k) = (i+1,k+1)
        else:
            C.append(B[j])
            (j,k) = (j+1,k+1)
    return(C)
```



So, that is divide and conquer. So, let us look at this merging thing in more detail. So, if I want to combine two sorted lists, and I want to write the actual code for this, then I have to look at these boundary conditions, which we came to at the end of the thing that we were executing. So, if one of the list is empty, we just copy the other list.

So, if I have run out of things in *A*, then I just take everything that is in *B* and put it at the end of *C*. Similarly, if everything in *B* is empty, then I just copy whatever is in *A* into *C*. And if they are both not empty, then I look at the first element that is visible, or the first element. I am scanning currently for both these lists. And I take the smaller of these two and append it to *C*. So, this is how merge works.

So, we repeat this until everything has been moved. So, this is the merge function, which we will quickly look at. So, we have two lists, and they could be a different length. So, *A* is of length *m* and *B* is of length *n*. So, now, what I will do is I will have my three lists, I have *A* and *B*, which are of length *m* and *n* and I have my *C*.

Now at any given point, I am going to be looking at some element here which I call *i* some element here, which I call *j* and some element here, which I will call *k*. So, as I progress, I am walking down these three lists the two lists that I am scanning to examine and the third list *C* which I am building up. So, I have these three indices *i*, *j* and *k* which are 0 initially and add this new list *C* which is empty. So, this is what I need to do to manipulate.

(Refer Slide Time: 9:39)

Merging sorted lists

- Combine two sorted lists *A* and *B* into *C*
 - If *A* is empty, copy *B* into *C*
 - If *B* is empty, copy *A* into *C*
 - Otherwise, compare first elements of *A* and *B*
 - Move the smaller of the two to *C*
 - Repeat till all elements of *A* and *B* have been moved

```
def merge(A,B):  
    (m,n) = (len(A),len(B))  
    (C,i,j,k) = ([],0,0,0)  
    while k < m+n:  
        if i == m:  
            C.extend(B[j:])  
            k = k + (n-j)  
        elif j == n:  
            C.extend(A[i:])  
            k = k + (m-i)  
        elif A[i] < B[j]:  
            C.append(A[i])  
            (i,k) = (i+1,k+1)  
        else:  
            C.append(B[j])  
            (j,k) = (j+1,k+1)  
    return(C)
```

Madhavan Mukund Merge Sort

So, now I go through these three steps. So, these three steps are here. So, if *A* is empty, so when is *A* empty when *i* have reached, *i* has reached the end. So, if *i* is equal to *m* that means *A* is empty. Then I just copy *B* to *C*. So, I use this Python function called `extend` which will take one list and extend it to with another list.


And what I am going to do is I am going to, in that process, move *k* by a whole lot. I am going to move *k* by however many letters I moved values I moved from *B*, which is it, was a *j* and I moved from *j* to *n*. So, *n* minus *j* elements have been shifted. If that is not the case, if *A* is not empty check if *B* is empty, in which case symmetrically I have *j* has gone to *n*.

In which case I extend *C* by *A* and I move *k* by *n* minus *i* should be sorry, *n* minus *i*. And finally, if neither of them is empty, then I compare *A*[*i*] and *B*[*j*] and if *A*[*i*] is smaller than *B*[*j*], then I append *A*[*i*] and I move *i* and *k* and otherwise, I append *B*[*j*] and I move *j* and *k*. So, if I push the indices, either *i* moves and *k* moves or *j* moves or *k* moves.

Now, when do I stop? Well, I have to eventually move *m* plus *n* elements into *k*, into *C*. So, *k* has to go from 0 to *n* plus *n*. So, long as *k* is not yet *m* plus *n*, I have work to do. But when *k* reaches *m* plus *n*, I am done. So, that becomes my terminating condition for this merge function.

(Refer Slide time: 11:12)


Merge sort



- To sort A into B , both of length n
- If $n \leq 1$, nothing to be done
- Otherwise
 - Sort $A[:n//2]$ into L
 - Sort $A[n//2:]$ into R
 - Merge L and R into B

```
def mergesort(A):  
    n = len(A)  
    if n <= 1:  
        return(A)  
    L = mergesort(A[:n//2])  
    R = mergesort(A[n//2:])  
    B = merge(L,R)  
    return(B)
```

$n/2-1$



Madhavan Mukund Merge Sort

So, having got the merge function, what does Merge Sort look like? Actually, it is much simpler than merge as you would expect. So, if you have a list, which has at most one element, there is nothing to be done. Otherwise, you sort the first half or the second half and merge them. So, this is what it is, you find the length of the list you want to do, if the list, length of the list is one or less, then you just return the list that you got saying is already sorted, otherwise you recursively call merge sort on the first half and the second half.

Now remember that this will stop at n by 2 minus 1. So, the lists will be disjoint, but which will cover the entire thing nothing will be left out. So, 0 to n by 2 minus 1 will be n part L and n by 2 to n the n will be in part r . Now having got both of these, I just apply my merge function to get a list which combines the two into a new list B and I return B . So, this is Merge Sort.

(Refer Slide Time: 12:12)

Summary



- Merge sort using divide and conquer to sort a list
- Divide the list into two halves
- Sort each half
- Merge the sorted halves
- Next, we have to check that the complexity is less than $O(n^2)$



To summarize, Merge Sort uses divide and conquer two sorted lists. We divide the list into two halves so what each half and merge the sorted halves. So, why did we come to this at all is because we said that insertion sort and selection sort were not acceptable in terms of their complexity because they were order n squared and we wanted to get something which is better than order n squared. So, to complete our discussion of merge sort, we have to analyze it which will be a separate exercise.