Merge Sort

CSE 373 - Data Structures

May 10, 2002

Readings and References

- Reading
  - › Section 7.6, *Data Structures and Algorithm Analysis in C*, Weiss
- Other References

“Divide and Conquer”

- Very important strategy in computer science:
  - › Divide problem into smaller parts
  - › Independently solve the parts
  - › Combine these solutions to get overall solution
- **Idea**: Divide array into two halves, *recursively* sort left and right halves, then *merge* two halves → known as Mergesort

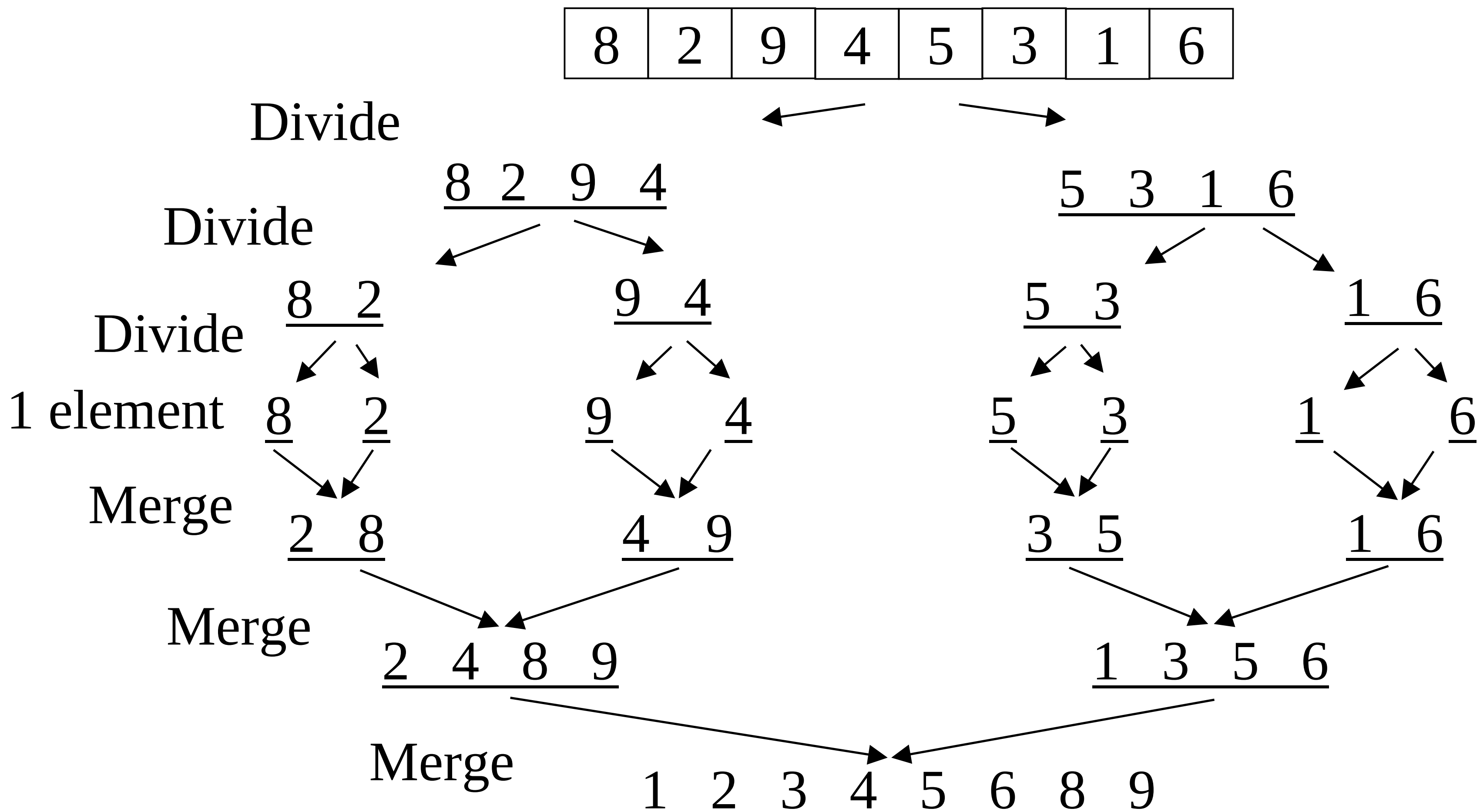
“Divide and Conquer”

- Example: Mergesort the input array:

0	1	2	3	4	5	6	7
8	2	9	4	5	3	1	6

- Divide it in two at the midpoint
- Conquer each side in turn (by recursively sorting)

Mergesort Example



Mergesort - driver

```
void Mergesort(ElementType A[], int N) {
  ElementType *TmpArray;
  TmpArray = malloc(N*sizeof(ElementType))
  FatalErrorMemory(TmpArray);
  MSort(A,TmpArray,0,N-1);
  free(TmpArray);
}
```

- Driver routine Mergesort calls the actual recursive implementation routine MSort with appropriate parameters
  - › Hides implementation details from outside callers

Mergesort - recursion

```
void MSort(ElementType A[], ElementType TmpArray[], int
   Left, int Right) {
   int Center;
   if (Left < Right) {
        Center = (Left+Right)/2;
        MSort(A, TmpArray, Left, Center);
        Msort(A, TmpArray, Center+1, Right);
        Merge(A, TmpArray, Left, Center+1, Right);
   }
}
```

- Divide, and leave the conquering to Merge …
  - › note the base case Left==Right

Mergesort - do it

```
void Merge(ElementType A[], ElementType TmpArray[], int
   Lpos, int Rpos, int RightEnd) {
   int i, LeftEnd, NumElements, TmpPos;
   LeftEnd = Rpos-1;
   TmpPos = Lpos;
   NumElements = RightEnd - Lpos + 1;
   while (Lpos <= LeftEnd && Rpos <= RightEnd)
        if (A[Lpos]<=A[Rpos]) TmpArray[TmpPos++]=A[Lpos++];
        else TmpArray[TmpPos++]=A[Rpos++];
   while (Lpos <= LeftEnd) TmpArray[TmpPos++]=A[Lpos++];
   while (Rpos <= RightEnd) TmpArray[TmpPos++]=A[Rpos++];
   for (i=0; i<NumElements; i++,RightEnd--)
        A[RightEnd] = TmpArray[RightEnd];
}
```

Mergesort Example

Divide down to 1 element

Merge to TmpArray

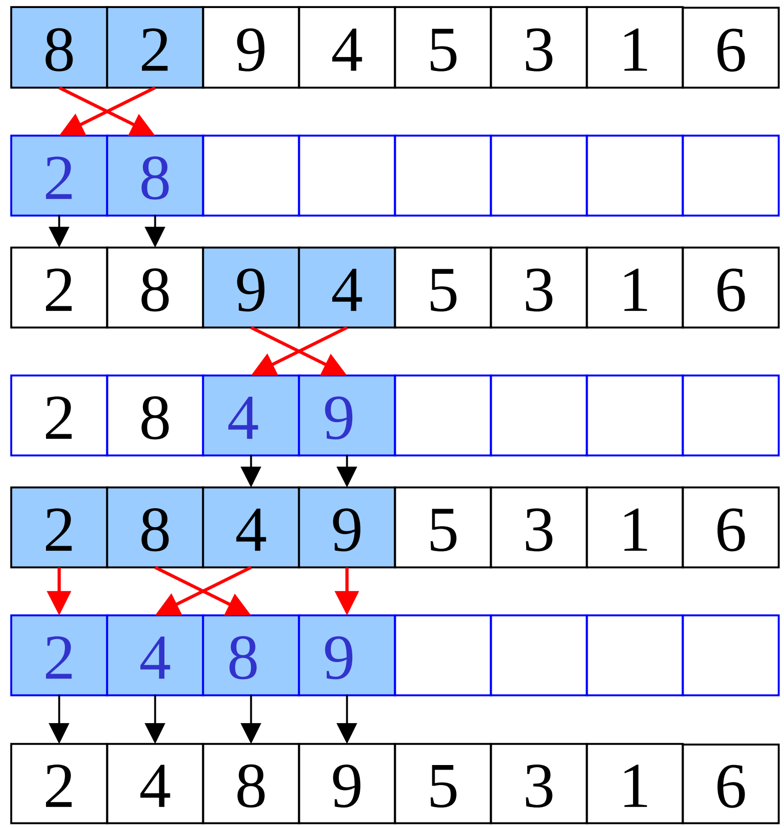
Copy back to A[]

Merge to TmpArray

Copy back to A[]

Merge to TmpArray

Copy back to A[]



Left half is now sorted ...

Mergesort Analysis

- Let T(N) be the running time for an array of N elements
- Mergesort divides array in half and calls itself on the two halves. After returning, it merges both halves using a temporary array
- Each recursive call takes T(N/2) and merging takes O(N)

Mergesort Recurrence Relation

- The recurrence relation for T(N) is:
  - › $T(1) = O(1)$
    - base case: 1 element array → constant time
  - › $T(N) = 2T(N/2) + N$
    - Sorting N elements takes
      - – the time to sort the left half
      - – plus the time to sort the right half
      - – plus an O(N) time to merge the two halves

Solving the Mergesort Recurrence Relation

- Solve the recurrence by expanding the terms:

  $T(N) = 2*T(N/2) + N$ and $T(N/2) = 2*T(N/4) + N/2$

  $T(N) = 2*[2*T(N/4) + N/2] + N$

  $= 2^2*T(N/2^2) + 2*N$

  $= 2^2[2*T(N/8) + N/4] + 2*N$

  $= 2^3*T(N/2^3) + 3*N$

  …

  $= 2^{\log N}*T(N/2^{\log N}) + (\log N)*N$ (recall that $2^{\log N} = N$)

  $= N * T(1) + N \log N$

  $= N * O(1) + N \log N = O(N \log N)$

  › T(N) is O(N log N)