CSC 212
Data Structures & Algorithms

Fall 2022 | Jonathan Schrader

Merge Sort

Housekeeping

Assignment 3

- Due Friday, 11:59p

Review Project [MEC]

- Due Next Friday, October 28, 11:59pm

Divide & Conquer

Divide the problem into smaller subproblems

Conquer recursively

- … each subproblem

Combine Solutions

Example

10	2	3	7	4	13	11	9

- sorting with insertion sort is n^2
- we can divide the array into two halves and sort them separately

2	3	7	10

4	9	11	13

- each subproblem could be sorted in $\approx \frac{n^2}{4}$
- sorting both halves will require $\approx 2\frac{n^2}{4}$ 🤔
- we need an additional operation to combine both solutions

Time "reduced" from $\approx n^2$ to $\approx \frac{n^2}{2} + n$

Merge Sort

Divide the array into two halves

- just need to calculate the midpoint

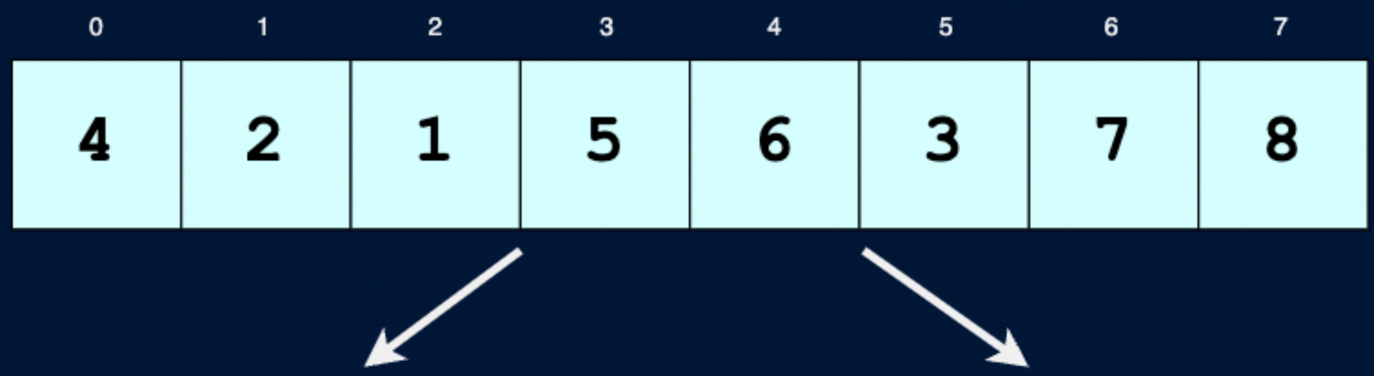
Conquer *Recursively* each half

- call Merge Sort on each half (i.e. solve 2 smaller problems)

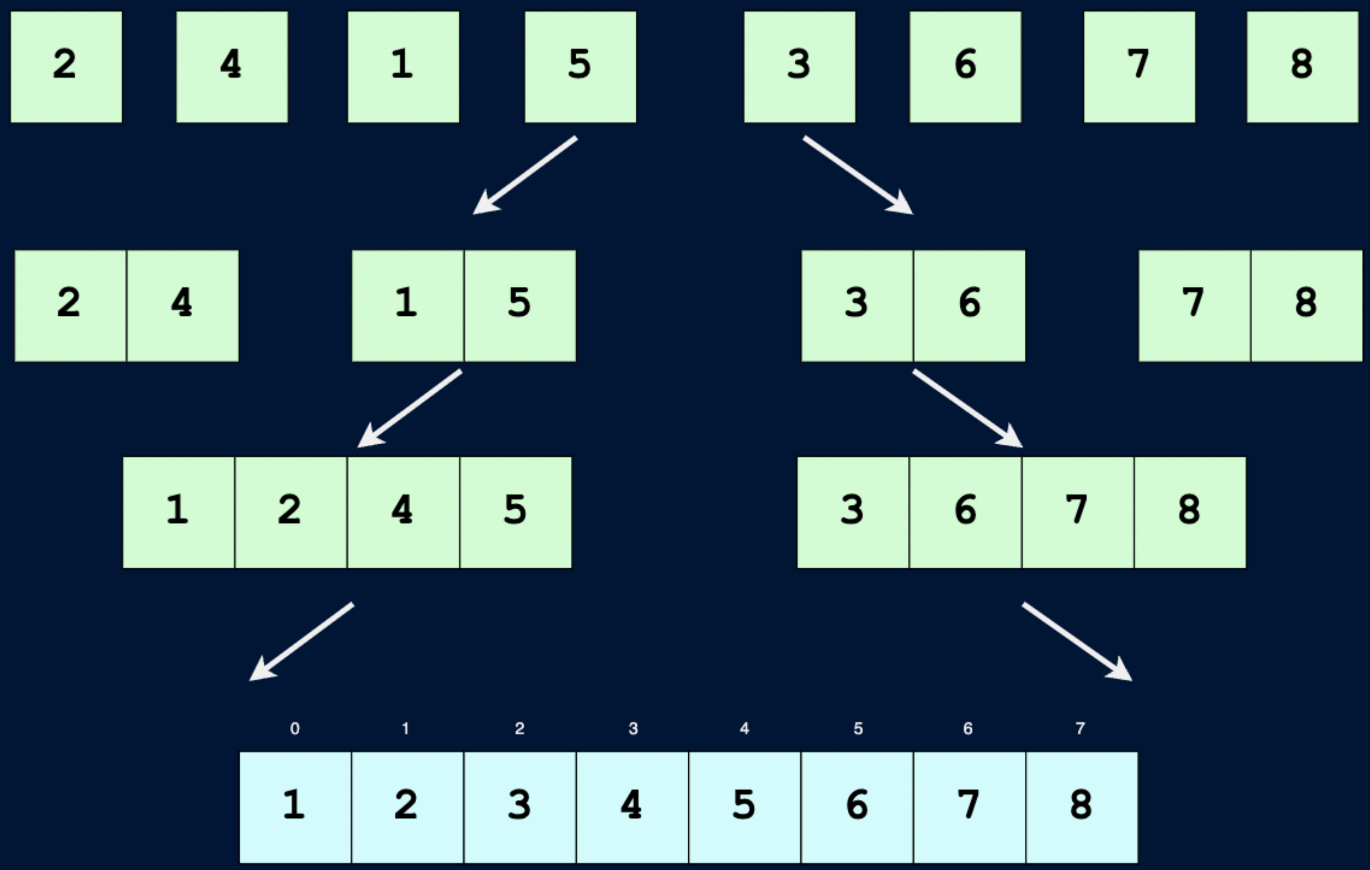
Merge Solutions

- after both calls are finished, proceed to *merge* the solutions

Divide…



...& Conquer



Merge Sort: pseudocode

```
if (hi <= lo)
  return;

int mid = lo + (hi - lo) / 2;

mergesort(A, lo, mid);

mergesort(A, mid + 1, hi);

merge(A, lo, mid, hi);
```

THINK BIG WE DO

Merge Sort

```
void r_mergesort(int *A, int *aux, int lo,int hi) {
  //basecase(single element or empty list)
  if (hi <= lo)
    return;

  //divide
  int mid = lo + (hi - lo) / 2;

  //recursively sort halves
  r_mergesort(A, aux, lo, mid);
  r_mergesort(A, aux, mid + 1, hi);

  //merge results
  merge(A, aux, lo, mid, hi);
}

void mergesort(int *A, int n) {
  int *aux = new int[n];
  r_mergesort(A, aux, 0, n - 1);
  delete[] aux;
}
```

Merging two sorted arrays

1	2	5	10	15

i

3	6	9	16	20

j

1	2	3	5	6	9	10	15	16	20

A secondary array is necessary

to guarantee a lineartime *operation*

Merge

```
void merge (int *A, int *aux, int lo, int mid,int hi) {
  // copy array
  std::memcpy(aux + lo, A + lo, (hi - lo + 1 * sizeof(A)));
  // merge
  int i = lo, j = mid + 1;
  for (int k = lo; k <= hi; k++) {
    if (i > mid)
      A[k]=aux[j++];
    else if (j > hi)
      A[k] = aux[i++];
    else if(aux[j] < aux[i])
      A[k] = aux[j++];
    else
      A[k] = aux[i++];
  }
}
```

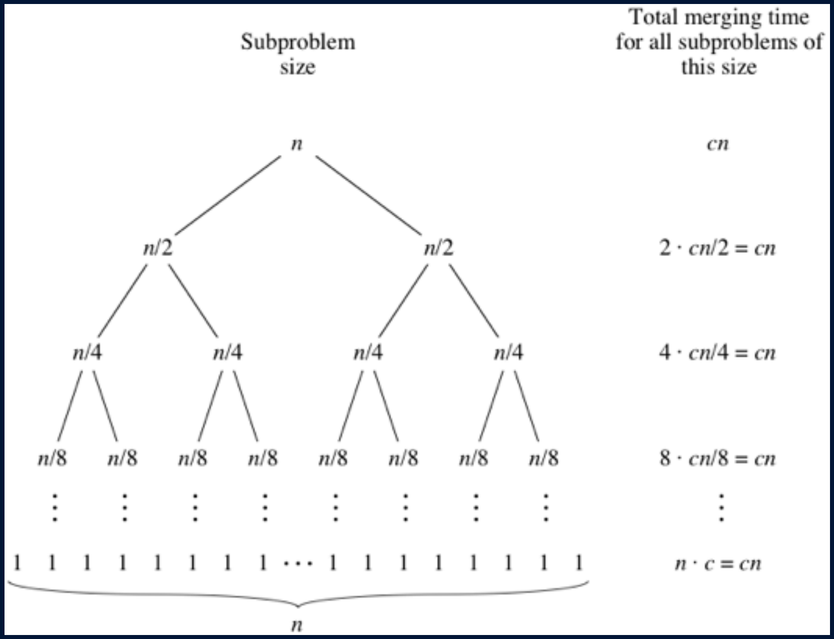
Analysis (recurrence)

```
void r_mergesort(int *A, int *aux, int lo,int hi) {
  //basecase(single element or empty list)
  if (hi <= lo)
    return;

  //divide
  int mid = lo + (hi - lo) / 2;

  //recursively sort halves
  r_mergesort(A, aux, lo, mid);
  r_mergesort(A, aux, mid + 1, hi);

  //merge results
  merge(A, aux, lo, mid, hi);
}

void mergesort(int *A, int n) {
  int *aux = new int[n];
  r_mergesort(A, aux, 0, n - 1);
  delete[] aux;
}
```

```
void merge (int *A, int *aux, int lo, int mid,int hi) {

  // copy array
  std::memcpy(aux + lo, A + lo, (hi - lo + 1 * sizeof(A)));
  // merge
  int i = lo, j = mid + 1;

  for (int k = lo; k <= hi; k++) {
    if (i > mid)
      A[k]=aux[j++];

    else if (j > hi)
      A[k] = aux[i++];

    else if(aux[j] < aux[i])
      A[k] = aux[j++];

    else
      A[k] = aux[i++];
  }
}
```

	Worst Case	Average Case	Best Case
Time Complexity	$O(n\ log\ n)$	$O(n\ log\ n)$	$O(n\ log\ n)$

Recursion Tree (trace)




```
void mergesort(int *A, int n) {
  int *aux = new int[n];

  r_mergesort(A, aux, 0, n - 1);

  delete[] aux;
}
```

```
void r_mergesort(int *A, int *aux, int lo,int hi) {
  if (hi <= lo) return;
  int mid = lo + (hi - lo) / 2;
  r_mergesort(A, aux, lo, mid);
  r_mergesort(A, aux, mid + 1, hi);
  merge(A, aux, lo, mid, hi);
}
```

Sorting Visualizer

Comments on Merge Sort

Major disadvantage

- it is *not* **in-place**
- in-place algorithm exists but it is complex and inefficient

Improvements

- use insertion sort for small arrays
  - avoid overhead on small instances (~10 elements)
- stop if already sorted
  - avoids unnecessary merge
  - works well with partially sorted arrays

In-place Sorting

Example

Think about reversing an array or string

solution 1: use an additional array of equal size

- what is the required extra memory?

solution 2: exchange first and last and work recursively on the inner part

- can do it iteratively as well
- what is the required extra memory?

In-place sorting

A sorting algorithm is *in-place* if it uses $O(log\ n)$ extra memory

Are selection and insertion sorts *in-place*?

```
void selectionSort(int arr[], int n)
{
    int i, j, min_idx;

    // One by one move boundary of
    // unsorted subarray
    for (i = 0; i < n-1; i++) {

        // Find the minimum element in
        // unsorted array
        min_idx = i;
        for (j = i+1; j < n; j++)
        if (arr[j] < arr[min_idx])
            min_idx = j;

        // Swap the found minimum element
        // with the first element
        if(min_idx!=i)
            swap(&arr[min_idx], &arr[i]);
    }
}
```

```
void insertionSort(int arr[], int n)
{
    int i, key, j;

    for (i = 1; i < n; i++)
    {
        key = arr[i];
        j = i - 1;

        // Move elements of arr[0..i-1],
        // that are greater than key, to one
        // position ahead of their
        // current position
        while (j >= 0 && arr[j] > key)
        {
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = key;
    }
}
```

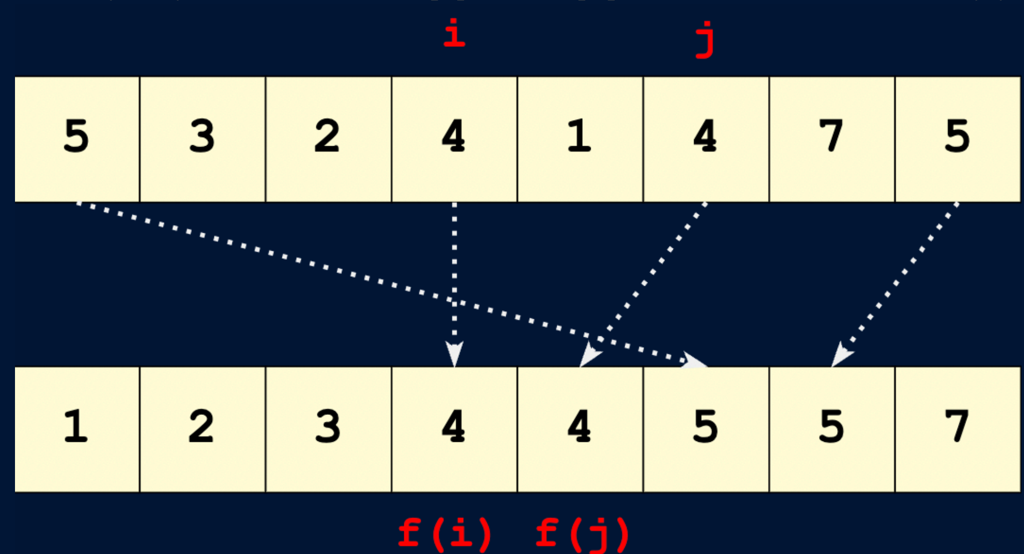
Stable Sorting

Stability

A sorting algorithm is *stable* if it preserves the order of equal elements

Consider sorting (in ascending order) a list A into a sorted list B. Let $f(i)$ be the index of element $A[i]$ in B. The sorting algorithm is stable if:

for any pair (i, j) such that $A[i] = A[j]$ and $i < j$, then $f(i) < f(j)$



Stable?

DL 2273	Detroit	5:30 am	Departed
WN 6240	Chicago - MDW	5:55 am	Departed
AA 489	Philadelphia	6:00 am	Departed
DL 1263	Atlanta	6:00 am	Departed
UA 6208	Washington - IAD	6:00 am	Departed
WN 1138	Baltimore	6:05 am	Departed
AA 5202	Washington - DCA	6:14 am	Departed
B6 475	Orlando	6:15 am	Departed
UA 4894	New York/Newark	6:15 am	Departed
AA 1703	Charlotte	6:17 am	Departed
WN 28	Orlando	6:55 am	Departed
AA 3410	Chicago - ORD	7:02 am	Departed
WN 6235	Tampa	7:05 am	Departed
UA 3615	Chicago - ORD	7:30 am	Departed
AA 1735	Philadelphia	8:02 am	Departed
AA 632	Charlotte	8:07 am	At 9:45 am
WN 6247	Fort Lauderdale	8:30 am	Departed
WN 2640	Washington - DCA	8:45 am	Departed
WN 3420	Chicago - MDW	8:45 am	Departed
AA 4280	Washington - DCA	8:49 am	At 10:20 am
WN 846	Baltimore	9:20 am	Departed
DL 305	Detroit	10:40 am	On time
AA 774	Philadelphia	10:51 am	On time
AA 1981	Charlotte	11:01 am	On time
WN 3020	Baltimore	11:20 am	On time
AA 5524	Washington - DCA	11:46 am	At 2:35 pm
AC 7379	Toronto	11:50 am	On time
AA 5550	Charlotte	11:54 am	On time
DL 5090	Detroit	12:32 pm	On time
WN 6296	Baltimore	12:35 pm	On time
DL 2225	Atlanta	12:48 pm	On time
AA 4424	Washington - DCA	1:38 pm	On time

sort,

then

sort

again

DL 1263	Atlanta	6:00 am	Departed
DL 2225	Atlanta	12:48 pm	On time
WN 1138	Baltimore	6:05 am	Departed
WN 846	Baltimore	9:20 am	Departed
WN 3020	Baltimore	11:20 am	On time
WN 6296	Baltimore	12:35 pm	On time
AA 632	Charlotte	8:07 am	At 9:45 am
AA 1703	Charlotte	6:17 am	Departed
AA 1981	Charlotte	11:01 am	On time
AA 5550	Charlotte	11:54 am	On time
WN 3420	Chicago - MDW	8:45 am	Departed
WN 6240	Chicago - MDW	5:55 am	Departed
AA 3410	Chicago - ORD	7:02 am	Departed
UA 3615	Chicago - ORD	7:30 am	Departed
DL 2273	Detroit	5:30 am	Departed
DL 305	Detroit	10:40 am	On time
DL 5090	Detroit	12:32 pm	On time
WN 6247	Fort Lauderdale	8:30 am	Departed
UA 4894	New York/Newark	6:15 am	Departed
B6 475	Orlando	6:15 am	Departed
WN 28	Orlando	6:55 am	Departed
AA 1735	Philadelphia	8:02 am	Departed
AA 489	Philadelphia	6:00 am	Departed
AA 774	Philadelphia	10:51 am	On time
WN 6235	Tampa	7:05 am	Departed
AC 7379	Toronto	11:50 am	On time
AA 4280	Washington - DCA	8:49 am	At 10:20 am
AA 5524	Washington - DCA	11:46 am	At 2:35 pm
AA 5202	Washington - DCA	6:14 am	Departed
WN 2640	Washington - DCA	8:45 am	Departed
AA 4424	Washington - DCA	1:38 pm	On time
UA 6208	Washington - IAD	6:00 am	Departed

Stability

Is selection sort stable?

- long distance swaps
- try: 5 2 3 8 4 5 6

Is insertion sort stable?

- equal items never pass each other (depends on correct implementation)

Sorting Algorithms

	Best-Case	Average-Case	Worst-Case	Stable	In-place
Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$	No	Yes
Insertion Sort	$O(n)$	$O(n^2)$	$O(n^2)$	Yes	Yes
Merge Sort	$O(n\ log\ n)$	$O(n\ log\ n)$	$O(n\ log\ n)$	Yes	No