USC Viterbi School of Engineering

Unit 4e

Sorting

Task 12a – From Unit 3d

- Find the maximum value in an array and move it to the end of the array
- Questions:
  - Do we scan through the array to find the maximum without moving it and swap it at the end ..or..
  - Do we move it as we can through the array

Find the maximum value and move it to the end of the array.

Index:	[0]	[1]	[2]	[3]	[4]	[5]	[6]
out	8	3	2	7	12	9	10

Task 12a

Find the maximum value and move it to the end of the array.

Index:	[0]	[1]	[2]	[3]	[4]	[5]	[6]	i	cmax
out	8	3						1	0

Index:	[0]	[1]	[2]	[3]	[4]	[5]	[6]	i	cmax
out	8		2					2	0

Index:	[0]	[1]	[2]	[3]	[4]	[5]	[6]	i	cmax
out	8			7				3	0

Index:	[0]	[1]	[2]	[3]	[4]	[5]	[6]	i	cmax
out					12			4	4

Index:	[0]	[1]	[2]	[3]	[4]	[5]	[6]	i	cmax
out					12	9		5	4

Index:	[0]	[1]	[2]	[3]	[4]	[5]	[6]	i	cmax
out					12		10	6	4

Index:	[0]	[1]	[2]	[3]	[4]	[5]	[6]	i	cmax
out					10	⟷	12	6	4

Task 12a

- What programming issues (mechanics) should you think about?
  - Do we just need to track the maximum VALUE or the INDEX of the maximum value?
  - Given that you can move the maximum number to the end of the array, how could this be used to SORT the entire array?

```
int main() {
  // setup array with data
  int n, val, data[100];
  cin >> n;
  for(int i=0; i < n; i++)
    { cin >> data[i]; }
  // now perform the given task






  // Print out results
  for(int i=0; i < n; i++){
    cout << data[i] << " ";
  }
  cout << endl;
  return 0;
}
```

Task 12b

Find the maximum value and move it to the end of the array.

Index:	[0]	[1]	[2]	[3]	[4]	[5]	[6]	i
out	8 ⇔	3						0

Index:	[0]	[1]	[2]	[3]	[4]	[5]	[6]	i
out	3	8 ⇔	2					1

Index:	[0]	[1]	[2]	[3]	[4]	[5]	[6]	i
out		2	8 ⇔	7				2

Index:	[0]	[1]	[2]	[3]	[4]	[5]	[6]	i
out			7	8	12			3

Index:	[0]	[1]	[2]	[3]	[4]	[5]	[6]	i
out					12 ⇔	9		4

Index:	[0]	[1]	[2]	[3]	[4]	[5]	[6]	i
out					9	12 ⇔	4	5

Index:	[0]	[1]	[2]	[3]	[4]	[5]	[6]
out	3	2	7	8	9	4	12

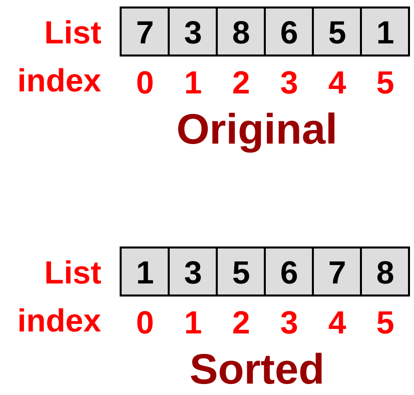
Task 12b

- What programming issues (mechanics) should you think about?
  - Do we just need to track the maximum VALUE or the INDEX of the maximum value?
  - Given that you can move the maximum number to the end of the array, how could this be used to SORT the entire array?

```
int main() {
  // setup array with data
  int n, val, data[100];
  cin >> n;
  for(int i=0; i < n; i++)
    { cin >> data[i]; }
  // now perform the given task









  // Print out results
  for(int i=0; i < n; i++){
    cout << data[i] << " ";
  }
  cout << endl;
  return 0;
}
```

Sorting

- Sorting requires us to move data around within an array
- Allows users to see and organize data more efficiently
- Behind the scenes it allows more effective searching of data
- There are MANY sorting algorithms out there, we will focus on two simple ones



Bubble Sort

- Main Idea: Keep comparing neighbors, moving larger item up and smaller item down until largest item is at the top. Repeat on list of size n-1
- Have one loop to count each pass, (a.k.a. i) to identify which index we need to stop at
- Have an inner loop start at the lowest index and count up to the stopping location comparing neighboring elements and advancing the larger of the neighbors

List	7	3	8	6	5	1

Original

List	3	7	6	5	1	8

After Pass 1

List	3	6	5	1	7	8

After Pass 2

List	3	5	1	6	7	8

After Pass 3

List	3	1	5	6	7	8

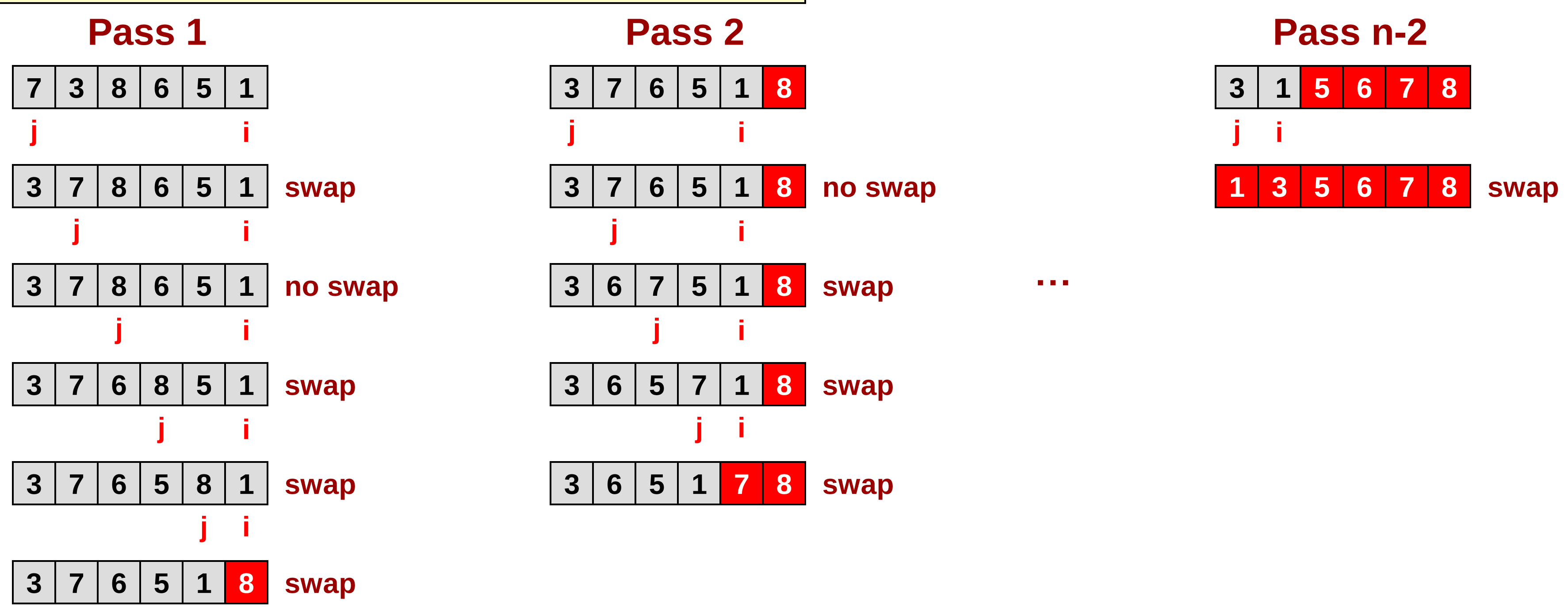
After Pass 4

List	1	3	5	6	7	8

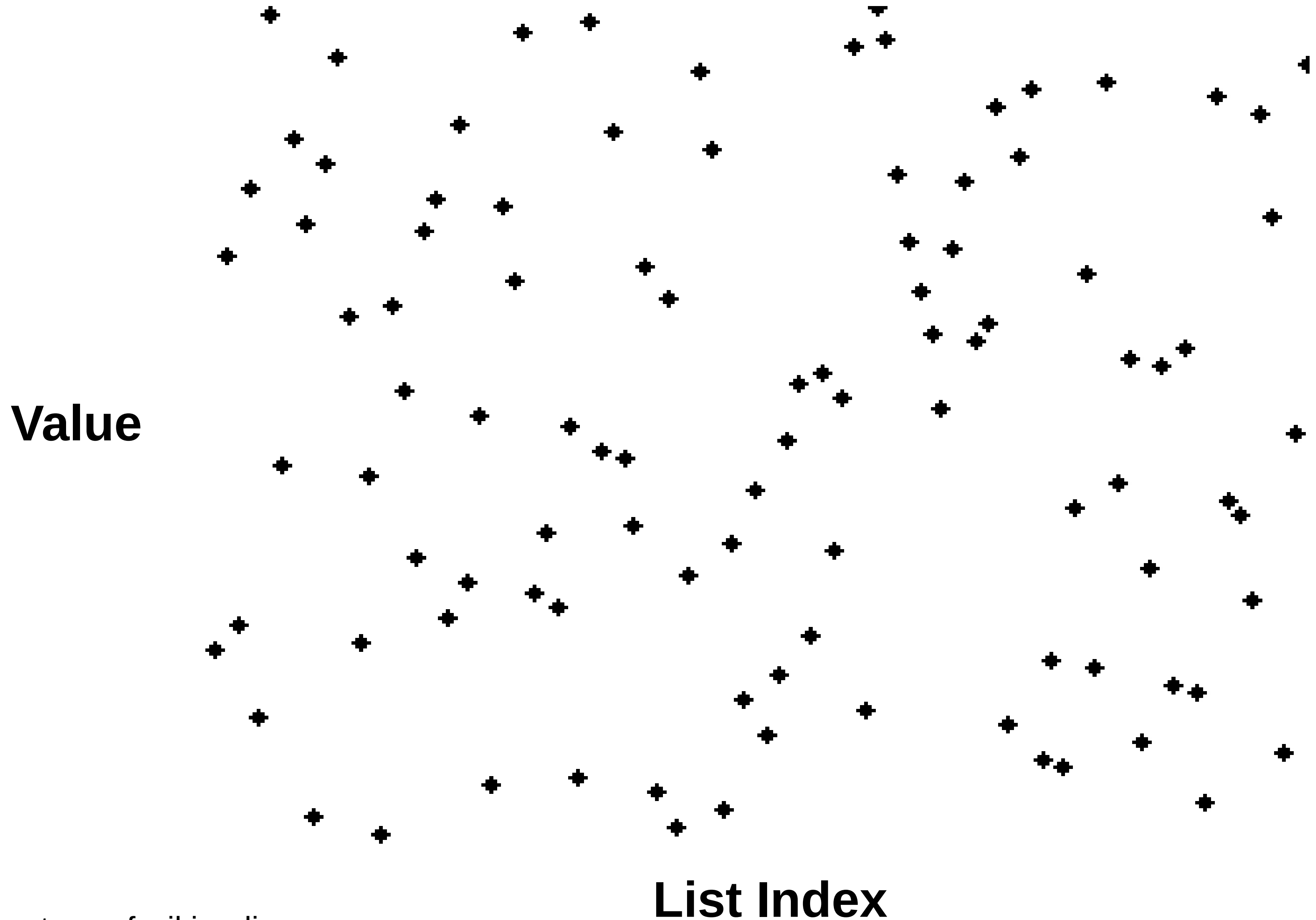
After Pass 5

Bubble Sort Algorithm

```
void bsort(int mylist[], int size)
{
  int i, j ;
  for(i=...     ){
     for(j=...   ){
        if(mylist[j] > mylist[j+1]) {
           // swap mylist[j] & mylist[j+1]
  }  }  }
}
```



Bubble Sort



Courtesy of wikipedia.org

Selection Sort

- Selection sort does away with the many swaps and just records where the min or max value is and performs one swap at the end
- The list/array can again be thought of in two parts
  - Sorted
  - Unsorted
- The problem starts with the whole array unsorted and slowly the sorted portion grows
- We could find the max and put it at the end of the list or we could find the min and put it at the start of the list
  - Just for variation let's choose the min approach

Selection Sort Algorithm

```
void ssort(int mylist[], int size)
{
  for(i=...){
     int min = i;
     for(j=... ){
        if(mylist[j] < mylist[min]) {
           min = j
     }  }
      // swap mylist[i] & mylist[min]
}
```

Pass 1 min=0

7	3	8	6	5	1	min=1
i	j					

7	3	8	6	5	1	min=1
i		j				

7	3	8	6	5	1	min=1
i			j			

7	3	8	6	5	1	min=1
i				j		

7	3	8	6	5	1	min=5
i					j	

1	3	8	6	5	7	swap

Pass 2 min=1

1	3	8	6	5	7	min=1
	i	j				

1	3	8	6	5	7	min=1
	i		j			

1	3	8	6	5	7	min=1
	i			j		

1	3	8	6	5	7	min=1
	i				j	

1	3	8	6	5	7	swap

Pass n-2 min=4

1	3	5	6	7	8	min=4
				i	j	

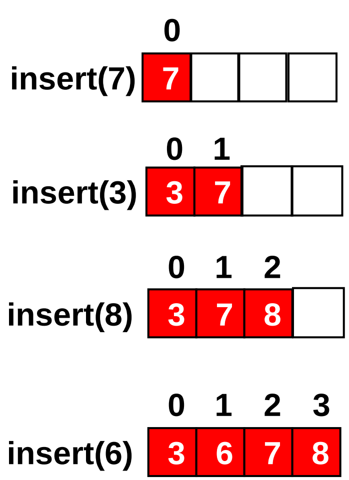
Selection Sort

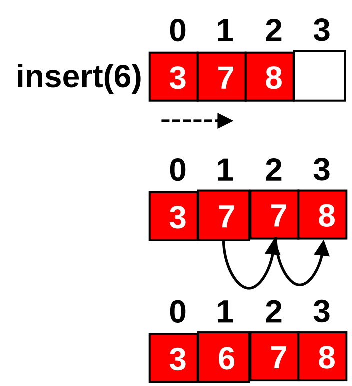
Courtesy of wikipedia.org

OPERATIONS ON A SORTED ARRAY

Insertion to a Sorted Array

- Another option rather than sorting an unordered array us to always insert new data into the correct location of the array
- See example below
- To insert, we must
  - Iterate until we find the appropriate location to place the new value
  - Make room for the new value by shifting the remaining items back a spot

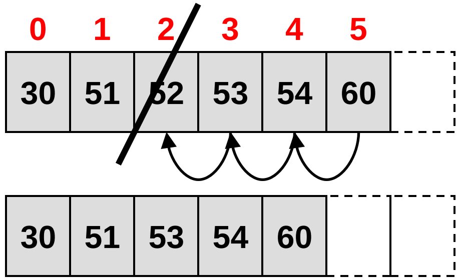




Removing from a Sorted Array

- Erasing / removing item at any location other than the very last item requires us to copy all items behind the removed item to the previous slot

To delete/remove the item at location 2 requires us to move everyone else up



USC Viterbi School of Engineering

COMPLEXITY & RUNTIME

Time Complexity

- Coming up with AN algorithm to solve a problem is often not TOO hard
- Coming up with a GOOD algorithm to solve a problem can be a bit harder
- We need a way to judge how "GOOD" an algorithm is
  - For us "GOOD" will mean how long the algorithm takes to solve the problem
  - We will count steps of work and come up with an answer in terms of n, where n is the size of the input/problem

Bubble Sorting

- Recall the bubble sort
- How much work do our nested loops require us to do
  - Think of each step/iteration as 1 unit of time/work

List	7	3	8	6	5	1

Original List is length N (N=6 for this example)

List	7	3	8	6	5	1

Original

List	3	7	6	5	1	8

Pass 1 (______ steps)

List	3	6	5	1	7	8

Pass 2 (______ steps)

List	3	5	1	6	7	8

Pass 3 (______ steps)

List	3	1	5	6	7	8

Pass 4 (______ steps)

List	1	3	5	6	7	8

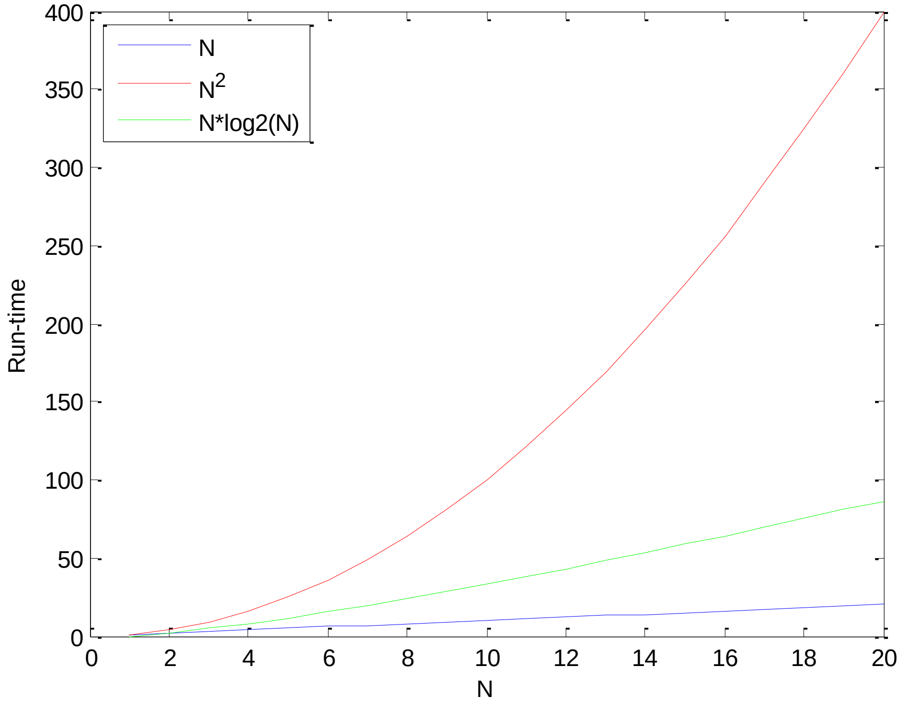
Pass 5 (______ steps)

List	1	3	5	6	7	8

Pass 6 (______ steps)

Complexity of Sort Algorithms

- Bubble Sort & Selection Sort
  - 2 Nested Loops
  - Execute outer loop n times
  - For each outer loop iteration, inner loop runs i times.
  - Time complexity is proportional to n^2
- Other sort algorithms can run in time proportional to: $n * \log_2(n)$



Importance of Time Complexity

- It makes the difference between effective and impossible
- Many important problems currently can only be solved with exponential run-time algorithms (e.g. $O(2^n)$ time)
- Usually algorithms are only practical if they run in polynomial time (e.g. O(n) or $O(n^2)$ etc.)

N	O(1)	$O(\log_2 n)$	O(n)	$O(n*\log_2 n)$	$O(n^2)$	$O(2^n)$
2	1	1	2	2	4	4
20	1	4.3	20	86.4	400	1,048,576
200	1	7.6	200	1,528.8	40,000	1.60694E+60
2000	1	11.0	2000	21,931.6	4,000,000	#NUM!

SOLUTIONS

Task 12a - Sol

- What programming issues (mechanics) should you think about?
  - Do we just need to track the maximum VALUE or the INDEX of the maximum value?
  - Given that you can move the maximum number to the end of the array, how could this be used to SORT the entire array?
    - Repeat the process for the first n-1 elements, then repeat for the first n-2 elements, etc.

```
int main() {
  // setup array with data
  int n, val, data[100];
  cin >> n;
  for(int i=0; i < n; i++)
    { cin >> data[i]; }
  // now perform the given task
  int cmax = 0;
  for(int i=1; i < n; i++) {
    if(data[i] > data[cmax]){
       cmax = i;
    }
  }
  // swap the max and end element
  int temp = data[n-1];
  data[n-1] = data[cmax];
  data[cmax] = temp;
  // Print out results
  for(int i=0; i < n; i++){
    cout << data[i] << " ";
  }
  cout << endl;
  return 0;
}
```

Task 12b - Sol

- What programming issues (mechanics) should you think about?
  - Do we just need to track the maximum VALUE or the INDEX of the maximum value?
  - Given that you can move the maximum number to the end of the array, how could this be used to SORT the entire array?
    - Repeat the process for the first n-1 elements, then repeat for the first n-2 elements, etc.

```
int main() {
  // setup array with data
  int n, val, data[100];
  cin >> n;
  for(int i=0; i < n; i++)
    { cin >> data[i]; }
  // now perform the given task
  for(int i=0; i < n-1; i++) {
    if(data[i] > data[i+1]){
       int temp = data[i];
       data[i] = data[i+1];
       data[i+1] = temp;
    }
  }
  // Print out results
  for(int i=0; i < n; i++){
    cout << data[i] << " ";
  }
  cout << endl;
  return 0;
}
```