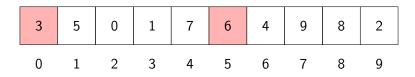
Class 18



elements 0 and 5 — are they in relative order with each other? yes, so move on



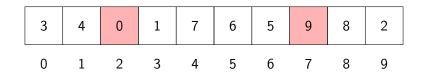
look at elements 1 and 6 — are they in order?

no, so order them using insertion sort on this pair



look at elements 1 and 6 — are they in order?

no, so order them using insertion sort on this pair



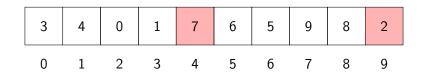
repeat with elements 2 and 7 — are they in order?

yes, keep going



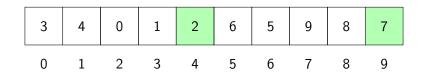
repeat with elements 3 and 8 — are they in order?

yes, keep going



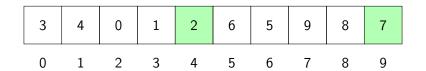
repeat with elements 4 and 9 — are they in order?

no, so order them (insertion sort)



repeat with elements 4 and 9 — are they in order?

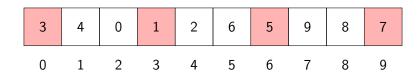
no, so order them (insertion sort)



repeat with elements 4 and 9 — are they in order?

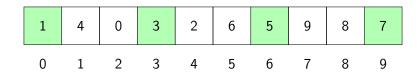
no, so order them (insertion sort)

at this point, the array is 5-sorted



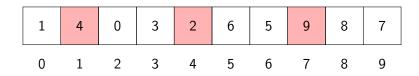
start over with element 0 and every third subsequent element — are they in order?

no, so order them with insertion sort



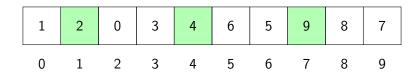
start over with element 0 and every third subsequent element — are they in order?

no, so order them with insertion sort



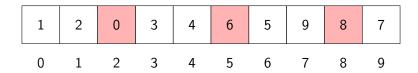
now element ${\bf 1}$ and every third subsequent element — are they in order?

no, so order them

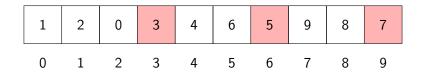


now element ${\bf 1}$ and every third subsequent element — are they in order?

no, so order them

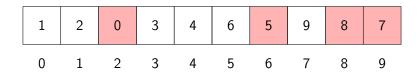


now element 2 and every third one after it — are they in order? yes, so no change



now element 2 and every third — are they in order?

again no change — the array is now 3-sorted

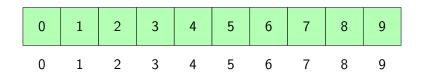


finally, look at the entire array — is it sorted?

no: the 0, 5, 7, and 8 are out of order

so apply normal insertion sort on the entire array

the array is now sorted



finally, look at the entire array — is it sorted?

no: the 0, 5, 7, and 8 are out of order

so apply normal insertion sort on the entire array

the array is now sorted

```
void shellsort(vector<unsigned>& array)
2
3
      size_t n = array.size();
4
     vector<size_t> gaps {5, 3, 1};
5
6
      for (auto gap : gaps)
7
8
       for (size_t i = gap; i < n; i++)
9
10
         unsigned temp = array.at(i);
11
         size_t j = i;
12
         while (j >= gap && temp < array.at(j - gap))</pre>
13
14
           array.at(j) = array.at(j - gap);
15
           j -= gap;
16
17
18
         array.at(j) = temp;
19
20
21
```

- named for Donald Shell
- B.S. chemical engineering from Michigan Tech in 3 years with the highest GPA in the school's history (record lasted 30 years)
- M.S. and Ph.D. in mathematics
- 1959 published the Shell sort algorithm
- the first known sorting algorithm to break the $O(n^2)$ barrier
- input size is # of array elements
- assume array of gaps is small and finite
- best and worst cases?

- named for Donald Shell
- B.S. chemical engineering from Michigan Tech in 3 years with the highest GPA in the school's history (record lasted 30 years)
- M.S. and Ph.D. in mathematics
- 1959 published the Shell sort algorithm
- the first known sorting algorithm to break the $O(n^2)$ barrier
- input size is # of array elements
- assume array of gaps is small and finite
- best and worst cases? yes the while loop

- named for Donald Shell
- B.S. chemical engineering from Michigan Tech in 3 years with the highest GPA in the school's history (record lasted 30 years)
- M.S. and Ph.D. in mathematics
- 1959 published the Shell sort algorithm
- the first known sorting algorithm to break the $O(n^2)$ barrier
- input size is # of array elements
- assume array of gaps is small and finite
- best and worst cases? yes the while loop
- best case:
- worst case:



- named for Donald Shell
- B.S. chemical engineering from Michigan Tech in 3 years with the highest GPA in the school's history (record lasted 30 years)
- M.S. and Ph.D. in mathematics
- 1959 published the Shell sort algorithm
- the first known sorting algorithm to break the $O(n^2)$ barrier
- input size is # of array elements
- assume array of gaps is small and finite
- best and worst cases? yes the while loop
- best case: linear: while loop never runs
- worst case: quadratic: while loop runs n times



- but this analysis is simplistic
- in fact, analyzing shellsort is fiendishly difficult
- way beyond the scope of this class
- Shell proposed gap sizes of n/2, n/4, ..., n/n
- these are actually quite bad
- 1963 Hibbard proposed gaps of $2^k 1, 2^{k-1} 1, \dots, 7, 3, 1$
- this gives dramatically improved performance (why?)
- much research into improved gap sequences
- much research into tighter bounds analysis

Visualization

https://www.youtube.com/watch?v=ZZuD6iUe3Pc

Search a List

- search an unsorted list for an element
- analyze two strategies:
 - 1. search the unsorted list to see if the element exists
 - 2. sort the list first, then search to see if the element exists

Strategy 1	Strategy 2
(unsorted)	(sort first)

worst case best case

Search a List

- search an unsorted list for an element
- analyze two strategies:
 - 1. search the unsorted list to see if the element exists
 - 2. sort the list first, then search to see if the element exists

	Strategy 1	Strategy 2
	(unsorted)	(sort first)
worst case	$T(n) \in O(n)$	$T(n) \in O(n \lg n)$
best case	$T(n) \in \Omega(1)$	$T(n) \in \Omega(n \lg n)$

Search a List

- search an unsorted list for an element
- analyze two strategies:
 - 1. search the unsorted list to see if the element exists
 - 2. sort the list first, then search to see if the element exists

	Strategy 1	Strategy 2
	(unsorted)	(sort first)
worst case	$T(n) \in O(n)$	$T(n) \in O(n \lg n)$
best case	$T(n) \in \Omega(1)$	$T(n) \in \Omega(n \lg n)$

• clearly, strategy 1 is superior

Multiple Runs

- now assume we are not going to do just one search
- assume we are going to do many (m) searches
- what is the analysis over m searches?

Strategy 1 Strategy 2 (unsorted) (sort first)

worst case best case

Multiple Runs

- now assume we are not going to do just one search
- assume we are going to do many (m) searches
- what is the analysis over *m* searches?

	Strategy 1 (unsorted)	Strategy 2 (sort first)
worst case	$T(m,n) \in O(mn)$	$T(m,n) \in O(m \lg n + n \lg n)$
best case	$T(m,n) \in \Omega(m)$	$T(m,n) \in \Omega(m+n \lg n)$

Multiple Runs

- now assume we are not going to do just one search
- assume we are going to do many (m) searches
- what is the analysis over *m* searches?

	Strategy 1 (unsorted)	Strategy 2 (sort first)
	(unsorted)	(SOLL HISL)
worst case best case	$T(m,n) \in O(mn)$ $T(m,n) \in \Omega(m)$	$T(m,n) \in O(m \lg n + n \lg n)$ $T(m,n) \in \Omega(m+n \lg n)$

• what value of m is the break-even point?

Presorting

- sometimes a job is easier if the values are known to be sorted
 - check if a value is unique
 - find the mode of a list of values
 - find the median of a list of values

Transform and Conquer

- a name for this strategy transform and conquer
- modify the arrangement of the data in the data structure to improve either the ease or speed of solution