SHELL SORT

SHELL SORT PARTITIONS THE ORIGINAL LIST INTO SUB-LISTS WHERE A SUB-LIST IS MADE OF ELEMENTS SEPARATED BY AN "INCREMENT"

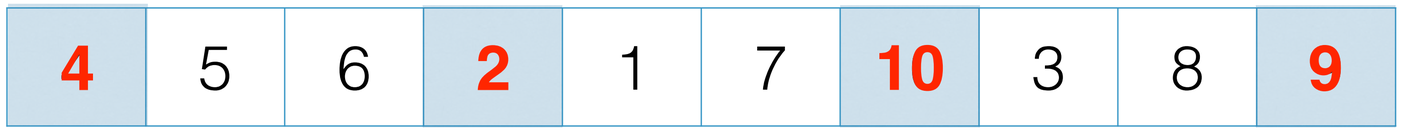
EACH SUB-LIST IS THEN SORTED USING INSERTION SORT

THE INCREMENT IS SLOWLY REDUCED TILL IT'S 1

AT THIS POINT IT'S BASICALLY INSERTION SORT OF A NEARLY SORTED LIST

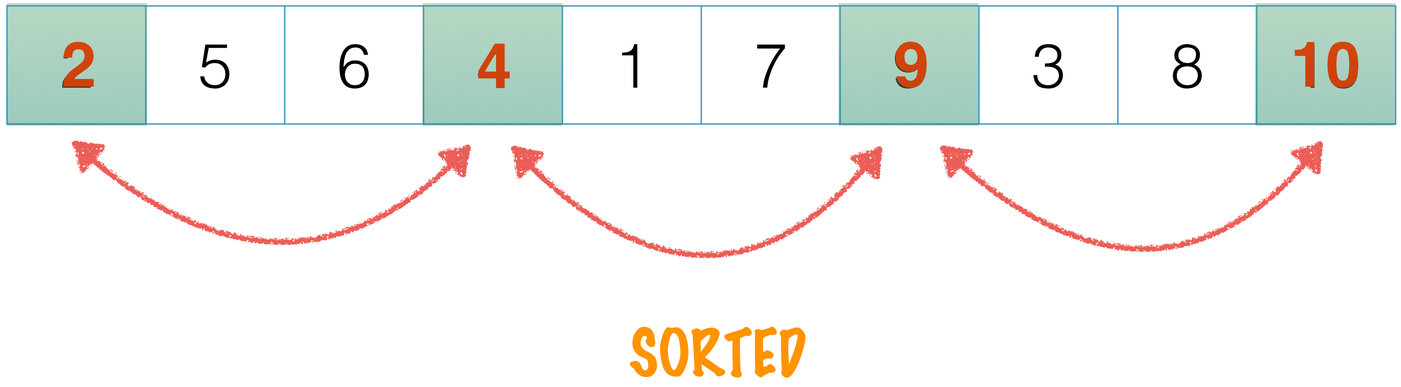
SHELL SORT

SUPPOSE THE INCREMENT = 3
THEN THE SUBLISTS WOULD
LOOK SOMETHING LIKE THIS



SHELL SORT

THE FIRST SUB-LIST FOR INCREMENT = 3 IS NOW SORTED

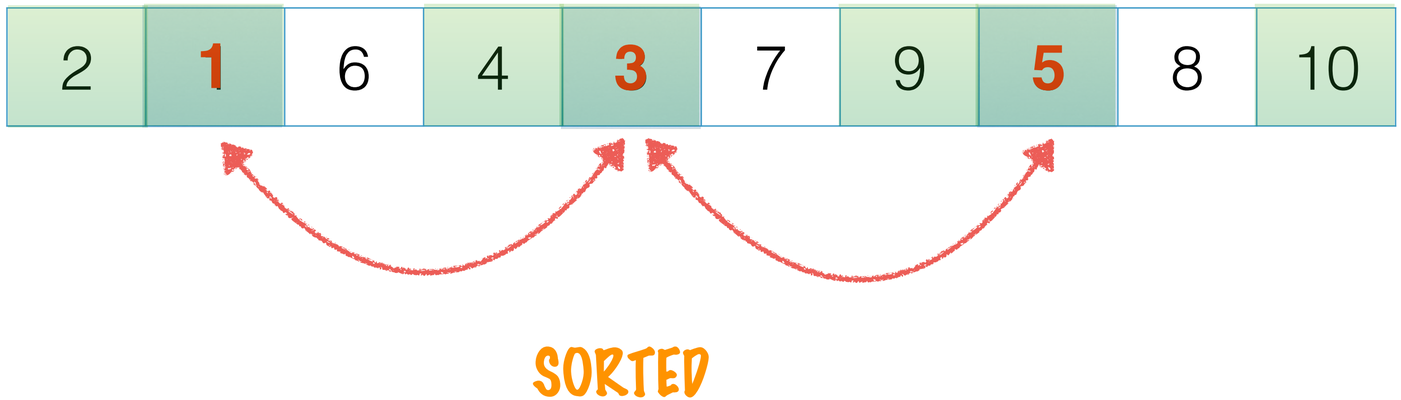


SHELL SORT

THE NEXT SUB-LIST
WITH INCREMENT
3 WOULD START
FROM INDEX 1

SHELL SORT

SORT THE NEXT
SUB-LIST



SHELL SORT

NOW THE LAST SUB-LIST
WITH INCREMENT 3

SHELL SORT

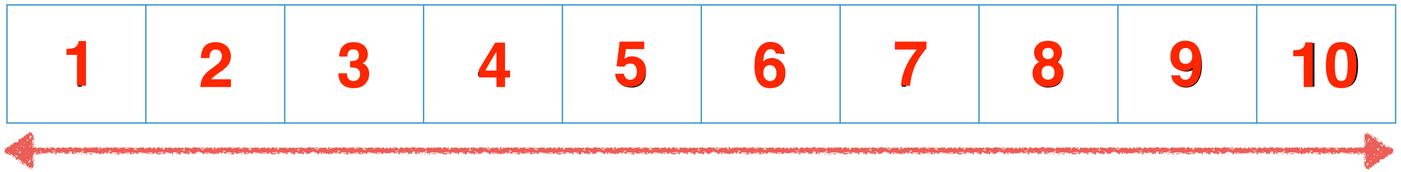
THE LIST IS ALMOST SORTED
NOW - ALL WE NEED IS
ANOTHER PASS WITH
INCREMENT = 1

SHELL SORT

2	1	6	4	3	7	9	5	8	10

SHELL SORT

SHELL SORT



THE COOL THING IS THAT SINCE THE LIST WAS ALMOST SORTED IT'S FAR EASIER TO GET TO A FULLY SORTED STATE WITH INCREMENT SET TO 1

MODIFIED INSERTION SORT - CODE

```
public static void insertionSort(int[] listToSort, int startIndex, int increment) {
    for (int i = startIndex; i < listToSort.length; i = i + increment) {
        for (int j = Math.min(i + increment, listToSort.length - 1);
             j - increment >= 0;
             j = j - increment) {
            if (listToSort[j] < listToSort[j - increment]) {
                swap(listToSort, j, j - increment);
            } else {
                break;
            }
            print(listToSort);
        }
    }
}
```

INSERTION SORT TAKES IN A START INDEX AND AN INCREMENT

ALL ITERATIONS THROUGH THE LIST ARE BASED ON THE INCREMENT

BREAK OUT OF THE INNER LOOP IF NO ELEMENT IS SWAPPED

ADJACENT ELEMENTS SEPARATED BY AN INCREMENT ARE COMPARED

SHELL SORT - CODE

```
public static void shellSort(int[] listToSort) {
    int increment = listToSort.length / 2;
    while (increment >= 1) {
        for (int startIndex = 0; startIndex < increment; startIndex++) {
            insertionSort(listToSort, startIndex, increment);
        }

        increment = increment / 2;
    }
}
```

WE'VE PICKED AN INCREMENT AT RANDOM

CALL INSERTION SORT ON ALL THE SUB-LISTS CREATED BY ELEMENTS "INCREMENT" APART

THE SORT IS COMPLETE WHEN INCREMENT REACHES 1

SLOWLY REDUCE THE INCREMENT

SHELL SORT USES INSERTION SORT, THE ENTIRE LIST IS DIVIDED AND THOSE SUB-LISTS ARE SORTED

GETTING THE EXACT COMPLEXITY OF SHELL SORT IS HARD BECAUSE IT DEPENDS ON THE INCREMENT VALUES CHOSEN

ALSO IT'S NOT CLEAR WHAT THE BEST INCREMENT VALUE IS

THE COMPLEXITY OF SHELL SORT IS BETTER THAN INSERTION SORT AS THE FINAL ITERATION WITH INCREMENT = 1 HAS TO WORK WITH A NEARLY SORTED LIST

THE COMPLEXITY OF SHELL SORT IS SOMEWHERE BETWEEN O(N) AND $O(N^2)$

DIFFERENT VALUES OF INCREMENTS PRODUCE DIFFERENT COMPLEXITIES

FOR INCREMENTS $2^K - 1$ FOR K = 1, 2, 3 ....

THE COMPLEXITY IS $O(N^{3/2})$

THE ALGORITHM IS ADAPTIVE SINCE ITS BASED ON INSERTION SORT WHICH IS ADAPTIVE

IT TAKES O(1) EXTRA SPACE, IT SORTS IN PLACE