Princeton Algorithms Course Part 1

Fouad Elbakly

1 Elementary Sorts

1.1 Selection sort

- Algorithm scans from left to right.
- Entries left of index are fixed and sorted
- Entries right of index are not fixes and are not sorted

1.1.1 Java Implementation

```
for (int i = 0; i < N; i++) {
 int min = i;
 for (int j = i + 1; j < N; j++)
  if (a[j] < a[min])
   min = j;
 swap(arr[i], arr[min]);
}
```

1.1.2 Time Complexity

Takes $O(n^2)$ time due to worst case scenario, outer loop N times, and first inner loop iteration N times.

1.2 Insertion Sort

- Scans from left to right
- Elements to the left of index are sorted
- Elements to the right of index have not been seen and are not sorted

1.2.1 Java Implementation

```
for (int i = 0; i < N; i++) {
 for (int j = i; j > 0; j--) {
  if (a[i] < a[j - 1])
   swap(a[j], a[j - 1])
  else
   break
```

```
 }
}
```

1.2.2 Time Complexity

Takes $O(n^2)$ time due to worst case scenario being a reversely ordered array. Thus, N iterations in outer loop and N iterations in inner loop.

1.3 Shellsort

- Moves entries more than one position at a time by h-sorting an array.
- When h is large, the number of subarrays is small, and as h gets smaller the array is nearly in order.
- Use the Knuth increment sequence $h = 3h + 1, (1, 4, 13, 40, 121, 364, ...)$

1.3.1 Java Implementation

```
int h = 1;

while (h < N/3)
 h = 3*h - 1;

while (h >= 1) {
 for (int i = h; i < N; i++)
  for (int j = i; j >= h && a[j] < a[j-h]; j -= h)
   swap(a[j], a[j - h]);
 h /= 3;
}
```

1.3.2 Time Complexity

Worst case time complexity is $O(n^2)$, but the average time complexity is unknown.

1.4 Shuffle Sort (Knuth Shuffle)

- Loop from left to right
- in interation i pick a random integer r between 0 and i.
- swap a[i] and a[r]

1.4.1 Java Implementation

```
int N = a.length;
for (int i = 0; i < N; i++) {
 int r = Random.nextInt(i + 1);
 swap(a[i], a[r]);
}
```

1.4.2 Time Complexity

Takes $O(n)$ time due to the need to loop through the array of size n only once.