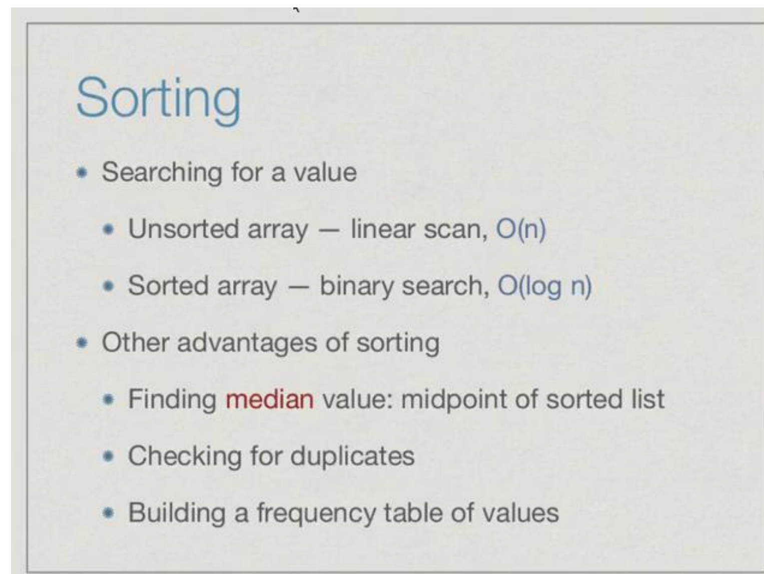
Programming, Data Structures and Algorithms in Python
Prof. Madhavan Mukund
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Week - 03
Lecture - 06
Selection Sort

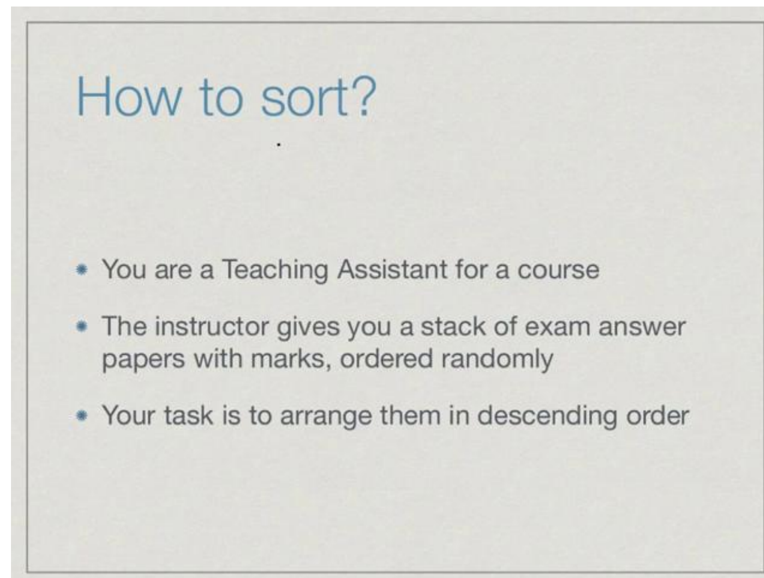
(Refer Slide Time: 00:02)



We have seen that searching becomes more efficient if we have a sorted sequence. So, for an unsorted array or a list, the linear scan is required and this takes order n time. However, if we have a sorted array we can use binary search and have the interval we half to search with each scan and therefore, take order log n time. Now sorting also gives us as a byproduct some other useful information. For instance, the median value - the median value in a set is a value such that half the value is a bigger and half are smaller.

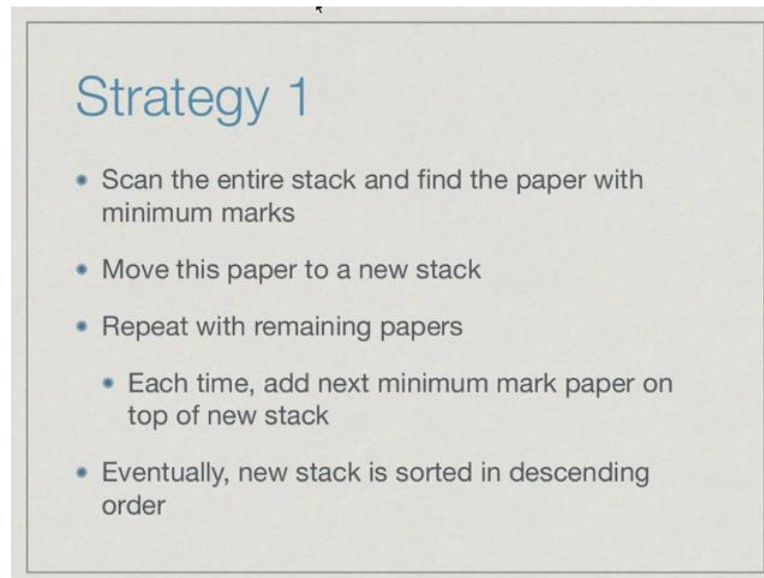
Once we have sorted a sequence, the midpoint automatically gives us the median. We can also do things like building frequency tables or checking for duplicates, essentially once we sort a sequence all identical values come together as a block. So, first of all by checking whether there is a block of size two, we can check whether there is a duplicate in our list; and for each block, if we count the size of the block, we can build a frequency table.

(Refer Slide Time: 01:06)



Let us look at some ways to sort sequences. So, forget about arrays and list for the moment, and let us think of sorting as a physical task to be performed. Suppose you are a teaching assistant for a course, and the teacher or the instructor has finished correcting the exam paper and now wants you to arrange them, so that the one with the largest marks - the highest marks is on top, the one with the second highest mark is below and so on. So, your task is to arrange the answer papers after correction in descending order of marks, the top most one should be the highest mark.

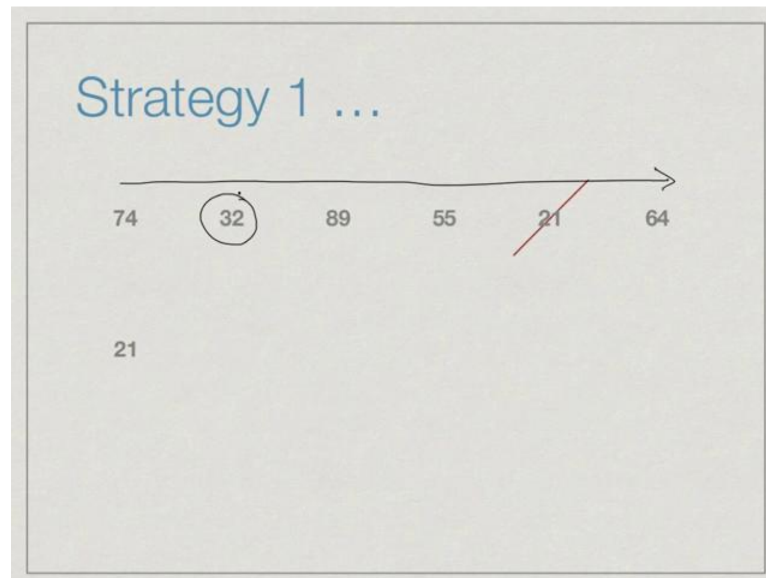
(Refer Slide Time: 01:46)



Here is one natural strategy to do this. So, what we can do is repeatedly look for the biggest or the smallest paper. Now in this case, we are going to build up the stack from the bottom, if the highest mark is on the top then the lowest mark will be at the bottom. So, what we do is we scan the entire stack, and find the paper with minimum marks. How do we do this, where we just keep looking at each paper in turn, each time we find a paper with the smaller mark then the one we have in our hand we change it and replace it by the one we have just found. At the end of the scan, in our hand we will have the paper with a minimum marks.

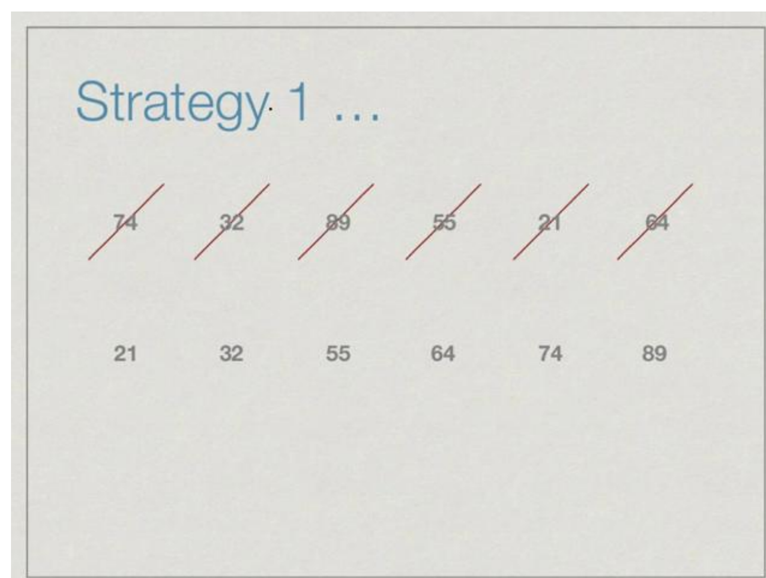
Initially, we assume that the top most paper has the minimum marks and we keep going down and replacing it with any lower mark we find. After this scan, we take the paper we have in our hand and put it aside and make a second stack where this is the bottom most thing. Now we have n minus 1 paper, we repeat the process. We look for the minimum mark amongst these n minus 1 papers and put this second lowest mark over all on top of the one we just put. Now, we have two papers stacked up, in order as we keep doing this we will build up the stack from bottom to top which has the lowest mark at the bottom, and the highest mark on the top.

(Refer Slide Time: 03:07)



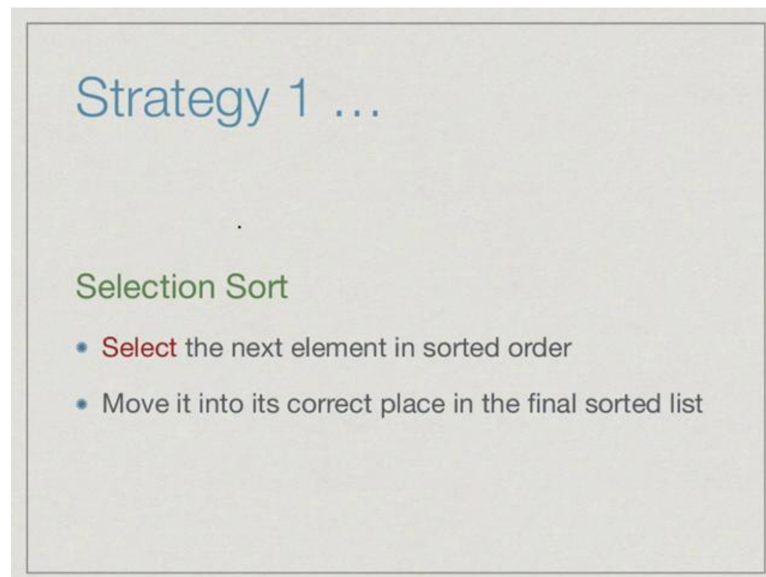
Suppose these are 6 papers. So, we have papers with mark 74, 32, 89, 55, 21 and 64. If we scan this list from left to right, then we will find that 21 is the lowest mark. So, our strategy says pick up the one with the lowest mark and move it to a new sequence or a new stack, so we do that.

(Refer Slide Time: 03:38)



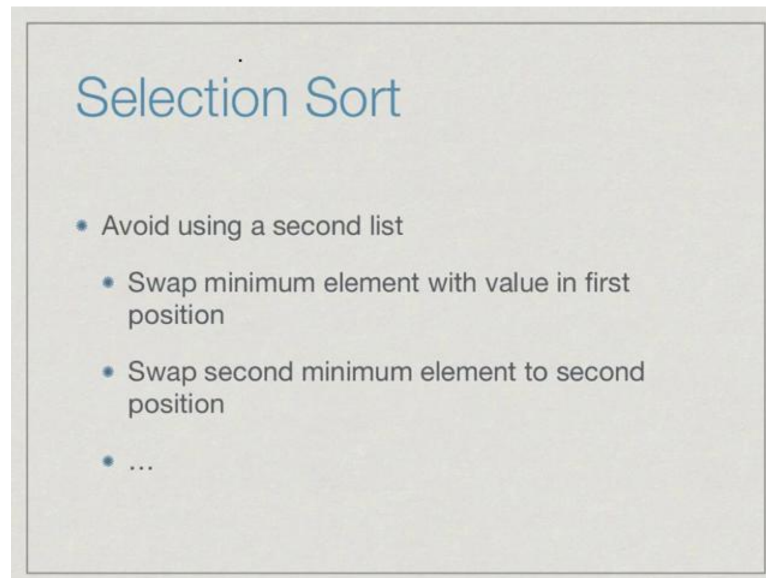
Now again, we scan from left to right this time of course 21 is gone, so we only have five numbers to scan. We will find that 32 is our next. And then proceeding in this way at the next step we will pick up 55 and then 64 and then 74, and finally 89. In this way by doing six scans on our list of six elements, we have build up a new sequence which has these six elements ordered according to their value.

(Refer Slide Time: 03:59)



This particular strategy which is very natural and intuitive has a name is called Selection Sort, because at each point we select the next element in sorted order and move it to the final sorted list which is in correct order.

(Refer Slide Time: 04:14)

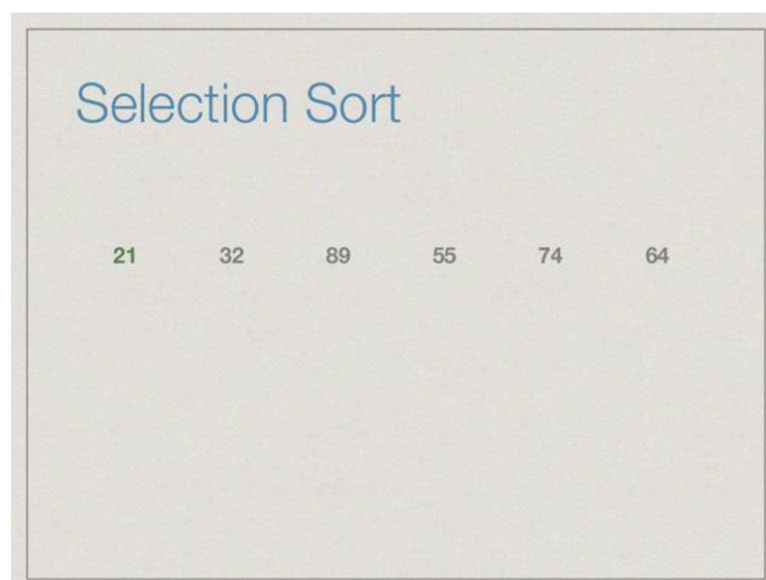


In the algorithm that we executed just now, we needed to build up a second list or a second sequence to store the sorted values. So, we kept pulling out things from the first sequence, and putting it in the second sequence. However, a little bit of thought will tell us that we do not need to do this. Whenever we pull out an element from the list as being the next smallest, we can move it to the beginning where it is supposed to be and exchange it with what is at the beginning. We can swap the minimum value with the value in the first position, after this we look at the second position onwards and find the second minimum value and swap it to the second position and so on.

(Refer Slide Time: 04:53)

So, if we were to execute this modified algorithm on the same input that we had before. In our first scan, we would start from the left in the first position is 74, and the minimum is at 21. Now, instead of moving 21 to a new list, we will now swap 21 and 74.

(Refer Slide Time: 05:09)



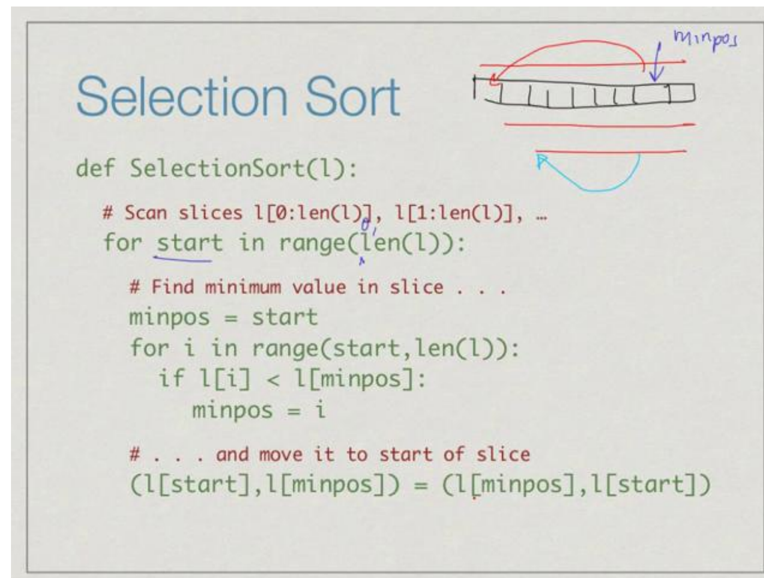
So, 21 comes in the beginning and 74 goes to the position where 21 was. Now we no

longer have to worry about anything to do with 21, we only need to look at this slice if you want to call it that starting from 32. We do this and we find the second smallest element. Now, the starting element is 32 and the second smallest element also happens to be 32 that is the smallest element in this slice. So, we just keep 32 where it is. Now we start the next slice from position two. The beginning element is 89 but the smallest element is 55. So, having finished this scan we would say 55 should move to the third position and 89 should replace it.

(Refer Slide Time: 05:58)

This way we just keep going on. Now we put 64 where 89 is, and finally 74 is in the correct place and 89 is also in the correct place. And we have a sorted sequence using selection sort where instead of making a second sequence, we have just systematically moved the smallest element we have found to the start with the segment or section that we are looking at right now.

(Refer Slide Time: 06:17)



Here is the very simple Python function which implements selection sort. The main idea about selection sort is that we have this sequence which has n elements to begin with. The first time, we will scan the entire sequence, and we would move this smallest element to this point. Then we will scan the sequence from one onwards, then we will scan the sequence on two onwards, and at each point in whichever segment where we are we will move the smallest element to the beginning.

We have this starting points of each scan, so the starting point initially starts at 0, and then it goes to 1, 2 up to the length of l minus 1. So, for the starting values from 0, implicitly this is 0 remember, 0 to the length of l minus 1, we first need to find the minimum value. We assume that the minimum value is at the beginning of that position of this slice. So we said the minimum position to be the starting position; remember the starting position is varying from 0 to the length of l minus 1.

So, each slice the starting position is the first position of the slice we have currently looking at. Then we scan from this position onwards and if we find a strictly smaller value. If l of i is smaller than what we correctly believe is the minimum value, we replace the minimum position by the current index. In this way after going through this entire thing, we would have found that say this position is the position of the minimum

value. Then we need to exchange these two, so we take the start position and the min position and we do this simultaneous walk, which we have seen before we take two values we exchange them using this pair notation.

(Refer Slide Time: 07:59)

Analysis of Selection Sort

- Finding minimum in unsorted segment of length k requires one scan, k steps
- In each iteration, segment to be scanned reduces by 1
- $T(n) = n + (n-1) + (n-2) + \ldots + 1 = n(n+1)/2 = O(n^2)$

$\frac{n^2}{2} + \frac{n}{2}$

Let us see how much time this takes. In each iteration or in each round of this, we are looking at a slice of length k, and we are finding the minimum in that and then exchanging it with the beginning. Now we have an unsorted sequence of values of length k, we have to look at all them to find the minimum value, because we have no idea where it is. We cannot stop at any point and declare that there are no smaller values beyond this. So, to find the minimum in an unsorted segment of length k, it requires one scan of k steps. And now we do this starting with the segment of the entire slice that is slice of length n then a slice of length n minus 1 and so on.

And so, if we write as usual T of n to be the time it takes for an input of size n to be sorted using selection sort this will be n for the first slice, n minus 1 for the second slice on I mean position one onwards, n minus 2 for the position two onwards and so on. And if I add this all up we have this familiar sum 1 plus 2 plus 3 up to n, which you will hopefully remember or you can look up is given by this expression n into n plus 1 by 2. Now n into n plus 1 by 2, if we expand it becomes n square by 2 plus n by 2.

Now this big O notation which tells us that it is proportional to n square; when we have expressions like this which have different terms like n, n square, n cube, it turns out that we only need to record the highest term. Since, n square is the highest term n square grows faster than n, we can simplify this to O n square. If you want to see why this is so, you should look up any standard algorithms book, it will explain to you how you calculate big O, but for our purposes it is enough to remember that big O just takes the highest term in the expression that we are looking at.

(Refer Slide Time: 09:57)

```
madhavan@dolphinair:...eek3/python/selectionsort$ more selectionsort.py
def SelectionSort(l):
  # Scan slices l[0:len(l)], l[1:len(l)], ...
  for start in range(len(l)):
    # Find minimum value in slice . . .  minpos = start
    for i in range(start,len(l)):
        minpos = start
        if l[i] < l[minpos]:
            minpos = i
            # .. and move it to start of slice
            (l[start],l[minpos]) = (l[minpos],l[start])
madhavan@dolphinair:...eek3/python/selectionsort$ python3.5
Python 3.5.2 (v3.5.2:4def2a2901a5, Jun 26 2016, 10:47:25)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> from selectionsort import *
>>> l = [3,7,2]
>>> SelectionSort(l)
>>> l
[2, 3, 7]
>>> l = list(range(500,0.-1))
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'float' object cannot be interpreted as an integer
>>> l = list(range(500,0,-1))
>>> ls
```

We said that for sorting algorithm like selection sort, which takes order n square will not work for the very large value say for length larger than about 5000. So, let us look at how this things works. First, this is the same code that we had in the slide, so selection sort scan slices from 0 up to the length of l minus 1. Let us start the Python interpreter. And now we will load selection sort from this file. Now notice the way selection sort works, it does not actually return a value that what selection sort does is it takes the value that the list that is passed to it and it sorts it in place.

In order to see anything from this, we have to first give it a name. So, let us take a list such as 3, 7, 2, for example, and say selection sort of l. And now we look at l, it is correctly sorted in the ascending order as 2, 3, and 7. Now in general we can take a

longer list. For instance, we can use this range function and say give me the list which is created by taking the range say from 500 to 0 with step of minus 1. So, this is an descending list of length 500.

(Refer Slide Time: 11:28)

```
449, 448, 447, 446, 445, 444, 443, 442, 441, 440, 439, 438, 437, 436, 435, 434, 433,
432, 431, 430, 429, 428, 427, 426, 425, 424, 423, 422, 421, 420, 419, 418, 417, 416,
415, 414, 413, 412, 411, 410, 409, 408, 407, 406, 405, 404, 403, 402, 401, 400, 399,
398, 397, 396, 395, 394, 393, 392, 391, 390, 389, 388, 387, 386, 385, 384, 383, 382,
381, 380, 379, 378, 377, 376, 375, 374, 373, 372, 371, 370, 369, 368, 367, 366, 365,
364, 363, 362, 361, 360, 359, 358, 357, 356, 355, 354, 353, 352, 351, 350, 349, 348,
347, 346, 345, 344, 343, 342, 341, 340, 339, 338, 337, 336, 335, 334, 333, 332, 331,
330, 329, 328, 327, 326, 325, 324, 323, 322, 321, 320, 319, 318, 317, 316, 315, 314,
313, 312, 311, 310, 309, 308, 307, 306, 305, 304, 303, 302, 301, 300, 299, 298, 297,
296, 295, 294, 293, 292, 291, 290, 289, 288, 287, 286, 285, 284, 283, 282, 281, 280,
279, 278, 277, 276, 275, 274, 273, 272, 271, 270, 269, 268, 267, 266, 265, 264, 263,
262, 261, 260, 259, 258, 257, 256, 255, 254, 253, 252, 251, 250, 249, 248, 247, 246,
245, 244, 243, 242, 241, 240, 239, 238, 237, 236, 235, 234, 233, 232, 231, 230, 229,
228, 227, 226, 225, 224, 223, 222, 221, 220, 219, 218, 217, 216, 215, 214, 213, 212,
211, 210, 209, 208, 207, 206, 205, 204, 203, 202, 201, 200, 199, 198, 197, 196, 195,
194, 193, 192, 191, 190, 189, 188, 187, 186, 185, 184, 183, 182, 181, 180, 179, 178,
177, 176, 175, 174, 173, 172, 171, 170, 169, 168, 167, 166, 165, 164, 163, 162, 161,
160, 159, 158, 157, 156, 155, 154, 153, 152, 151, 150, 149, 148, 147, 146, 145, 144,
143, 142, 141, 140, 139, 138, 137, 136, 135, 134, 133, 132, 131, 130, 129, 128, 127,
126, 125, 124, 123, 122, 121, 120, 119, 118, 117, 116, 115, 114, 113, 112, 111, 110,
109, 108, 107, 106, 105, 104, 103, 102, 101, 100, 99, 98, 97, 96, 95, 94, 93, 92, 91
, 90, 89, 88, 87, 86, 85, 84, 83, 82, 81, 80, 79, 78, 77, 76, 75, 74, 73, 72, 71, 70,
 69, 68, 67, 66, 65, 64, 63, 62, 61, 60, 59, 58, 57, 56, 55, 54, 53, 52, 51, 50, 49,
48, 47, 46, 45, 44, 43, 42, 41, 40, 39, 38, 37, 36, 35, 34, 33, 32, 31, 30, 29, 28, 2
7, 26, 25, 24, 23, 22, 21, 20, 19, 18, 17, 16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5,
 4, 3, 2, 1]
>>>
```

If I look at l, it is 500 down to 1.

(Refer Slide Time: 11:30)

```
 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87,
88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 10
7, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 12
4, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 14
1, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156, 157, 15
8, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 17
5, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 19
2, 193, 194, 195, 196, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207, 208, 20
9, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220, 221, 222, 223, 224, 225, 22
6, 227, 228, 229, 230, 231, 232, 233, 234, 235, 236, 237, 238, 239, 240, 241, 242, 24
3, 244, 245, 246, 247, 248, 249, 250, 251, 252, 253, 254, 255, 256, 257, 258, 259, 26
0, 261, 262, 263, 264, 265, 266, 267, 268, 269, 270, 271, 272, 273, 274, 275, 276, 27
7, 278, 279, 280, 281, 282, 283, 284, 285, 286, 287, 288, 289, 290, 291, 292, 293, 29
4, 295, 296, 297, 298, 299, 300, 301, 302, 303, 304, 305, 306, 307, 308, 309, 310, 31
1, 312, 313, 314, 315, 316, 317, 318, 319, 320, 321, 322, 323, 324, 325, 326, 327, 32
8, 329, 330, 331, 332, 333, 334, 335, 336, 337, 338, 339, 340, 341, 342, 343, 344, 34
5, 346, 347, 348, 349, 350, 351, 352, 353, 354, 355, 356, 357, 358, 359, 360, 361, 36
2, 363, 364, 365, 366, 367, 368, 369, 370, 371, 372, 373, 374, 375, 376, 377, 378, 37
9, 380, 381, 382, 383, 384, 385, 386, 387, 388, 389, 390, 391, 392, 393, 394, 395, 39
6, 397, 398, 399, 400, 401, 402, 403, 404, 405, 406, 407, 408, 409, 410, 411, 412, 41
3, 414, 415, 416, 417, 418, 419, 420, 421, 422, 423, 424, 425, 426, 427, 428, 429, 43
0, 431, 432, 433, 434, 435, 436, 437, 438, 439, 440, 441, 442, 443, 444, 445, 446, 44
7, 448, 449, 450, 451, 452, 453, 454, 455, 456, 457, 458, 459, 460, 461, 462, 463, 46
4, 465, 466, 467, 468, 469, 470, 471, 472, 473, 474, 475, 476, 477, 478, 479, 480, 48
1, 482, 483, 484, 485, 486, 487, 488, 489, 490, 491, 492, 493, 494, 495, 496, 497, 49
8, 499, 500]
>>>
```

And now if I say insertion uh selections sort of l, then it gets sorted as 1 to 500.

(Refer Slide Time: 11:44)

```
2, 193, 194, 195, 196, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207, 208, 20
9, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220, 221, 222, 223, 224, 225, 22
6, 227, 228, 229, 230, 231, 232, 233, 234, 235, 236, 237, 238, 239, 240, 241, 242, 24
3, 244, 245, 246, 247, 248, 249, 250, 251, 252, 253, 254, 255, 256, 257, 258, 259, 26
0, 261, 262, 263, 264, 265, 266, 267, 268, 269, 270, 271, 272, 273, 274, 275, 276, 27
7, 278, 279, 280, 281, 282, 283, 284, 285, 286, 287, 288, 289, 290, 291, 292, 293, 29
4, 295, 296, 297, 298, 299, 300, 301, 302, 303, 304, 305, 306, 307, 308, 309, 310, 31
1, 312, 313, 314, 315, 316, 317, 318, 319, 320, 321, 322, 323, 324, 325, 326, 327, 32
8, 329, 330, 331, 332, 333, 334, 335, 336, 337, 338, 339, 340, 341, 342, 343, 344, 34
5, 346, 347, 348, 349, 350, 351, 352, 353, 354, 355, 356, 357, 358, 359, 360, 361, 36
2, 363, 364, 365, 366, 367, 368, 369, 370, 371, 372, 373, 374, 375, 376, 377, 378, 37
9, 380, 381, 382, 383, 384, 385, 386, 387, 388, 389, 390, 391, 392, 393, 394, 395, 39
6, 397, 398, 399, 400, 401, 402, 403, 404, 405, 406, 407, 408, 409, 410, 411, 412, 41
3, 414, 415, 416, 417, 418, 419, 420, 421, 422, 423, 424, 425, 426, 427, 428, 429, 43
0, 431, 432, 433, 434, 435, 436, 437, 438, 439, 440, 441, 442, 443, 444, 445, 446, 44
7, 448, 449, 450, 451, 452, 453, 454, 455, 456, 457, 458, 459, 460, 461, 462, 463, 46
4, 465, 466, 467, 468, 469, 470, 471, 472, 473, 474, 475, 476, 477, 478, 479, 480, 48
1, 482, 483, 484, 485, 486, 487, 488, 489, 490, 491, 492, 493, 494, 495, 496, 497, 49
8, 499, 500]
>>> l = list(range(1000,0,-1))
>>> SelectionSort(l)
>>> SelectionSort(l)
>>> l = list(range(2000,0,-1))
>>> SelectionSort(l)
>>> l = list(range(5000,0,-1))
>>> SelectionSort(l)
```

Now our claim is that this will stop working effectively around 5000. So, let us see if I make this as 1000 instead of 500, and run selection sort then you can see there is an appreciable gap. Now if I do it for say 2000, then there is slightly longer gap. If I do it for 5000 then you can see it takes a little bit of time right it takes more than one second for sure. This is just to validate our claim that in Python if you expect to do something in one second then you better make sure that the number of steps is below about 10 to the 7. And since 5000 square takes you well beyond 10 to the 7, you can expect to take a very long time.