IE5600 Applied Programming | Week 2

Complexity, Divide and Conquer, and Greedy Method

Andrew Lim

1

1

Properties of an algorithm?

2

- Input
  - Zero or more quantities are externally supplied
- Output
  - At least one quantity is produced
- Definiteness
  - Each instruction is clear and unambiguous
- Finiteness
  - Terminates after a finite number of steps
- Effectiveness
  - Every instruction must be very basic so that it can be carried out in principle by a person using only pencil and paper. It not only need to be definite; it has to be feasible

2

Four Distinct Areas of Study

3

- How to devise algorithms
- How to validate algorithms
- How to analyze algorithms
- How to test algorithms

Devising My First Sorting Algorithm

4

Suppose we want to devise a program to sort a collection of $n \geq 1$ elements

First attempt – given as follows:

From those elements that are currently unsorted, find the smallest and place it next in the sorted list

Second attempt

```
# Python program for implementation of Selection Sort

# Traverse through all array elements
    # Find the minimum element in remaining
    # unsorted array

    # Swap the found minimum element with
    # the first element
```

Selection Sort

5

Third attempt

```
# Python program for implementation of Selection
# Sort


# Traverse through all array elements
for i in range(len(arr)):

    # Find the minimum element in remaining
    # unsorted array
    min_idx = i
    for j in range(i+1, len(arr)):
        if arr[min_idx] > arr[j]:
            min_idx = j

    # Swap the found minimum element with
    # the first element
    arr[i], arr[min_idx] = arr[min_idx], arr[i]
```


Generate Random Data Test

6

```
import random
data = [random.randint(1,65535) for i in range(100)]
print(data)
```

```
[36376, 16789, 6084, 63336, 64179, 65267, 53037, 33274, 35869, 9169, 6611, 9497, 37103, 54654, 30466, 6124, 38614, 38907, 11
543, 23708, 7594, 50812, 20214, 63605, 30940, 35948, 6751, 19225, 60350, 52406, 24112, 56616, 47729, 3238, 39874, 17978, 590
59, 44025, 58163, 6975, 40102, 21931, 38397, 44305, 28940, 35522, 47443, 19925, 20802, 56519, 21794, 19160, 21414, 64665, 23
78, 63429, 37064, 38115, 26844, 20387, 35347, 18108, 9712, 36343, 49117, 42666, 19355, 24607, 8287, 3288, 55350, 60608, 808,
55366, 30318, 11173, 46924, 62765, 45951, 42926, 4042, 6361, 43823, 40211, 13703, 41335, 56395, 38848, 57683, 19044, 2936, 2
2256, 10844, 24607, 34032, 10288, 33078, 2076, 22558, 26621]
```

```
import random
data = [random.randint(1,65535) for _ in range(100)]
print(data)
```

```
[42663, 35356, 9417, 21060, 2223, 65014, 21538, 47296, 18763, 38966, 14829, 14541, 59631, 26684, 62094, 42272, 34955, 39729,
59285, 38685, 5648, 1859, 41164, 64508, 34934, 63316, 14427, 35518, 3552, 64557, 4915, 41450, 39260, 11651, 57201, 7951, 279
79, 35379, 33464, 62711, 21360, 49081, 12692, 58194, 2452, 39507, 30247, 59572, 13231, 62416, 42454, 47061, 5721, 58966, 510
93, 50331, 59318, 60187, 14294, 26787, 48750, 39291, 13271, 12399, 44268, 61650, 59436, 54318, 8631, 28319, 20189, 17603, 13
750, 20844, 4959, 4024, 60683, 9403, 48310, 5606, 45001, 61061, 352, 27661, 27956, 7558, 2803, 38218, 1850, 2315, 59956, 615
88, 29963, 14875, 44033, 57218, 24989, 52696, 7524, 18783]
```

Run Selection Sort

7

```
def SelectionSort(arr):

    # Python program for implementation of Selection
    # Sort


    # Traverse through all array elements
    for i in range(len(arr)):

        # Find the minimum element in remaining
        # unsorted array
        min_idx = i
        for j in range(i+1, len(arr)):
            if arr[min_idx] > arr[j]:
                min_idx = j

        # Swap the found minimum element with
        # the first element
        arr[i], arr[min_idx] = arr[min_idx], arr[i]

print(data)
SelectionSort(data)
print(data)
```

```
[48057, 13305, 22166, 8749, 33542, 18447, 35455, 47422, 17244, 31839, 35804, 10231, 9333, 38779, 16633, 42706, 498, 18105, 12799, 29523]
[498, 8749, 9333, 10231, 12799, 13305, 16633, 17244, 18105, 18447, 22166, 29523, 31839, 33542, 35455, 35804, 38779, 42706, 47422, 48057]
```


Time Selection Sort

8

```
for i in [100,1000,2000,3000,4000,5000,6000,7000,8000,9000,10000,50000]:
    data = [random.randint(1,65535) for _ in range(i)]
    # Remember to import time
    s=time.time()
    SelectionSort(data)
    t=time.time()-s
    print("(Size="+str(i)+",Time="+str(t)+")", end='')
    print("(Size={},Time={:.4f})".format(i,t))

```

```
(Size=100,Time=0.0009965896606445312)(Size=100,Time=0.0010)
(Size=1000,Time=0.0678260326385498)(Size=1000,Time=0.0678)
(Size=2000,Time=0.20245599746704102)(Size=2000,Time=0.2025)
(Size=3000,Time=0.4578282833099365)(Size=3000,Time=0.4578)
(Size=4000,Time=0.8218302726745605)(Size=4000,Time=0.8218)
(Size=5000,Time=1.232703685760498)(Size=5000,Time=1.2327)
(Size=6000,Time=1.799309253692627)(Size=6000,Time=1.7993)
(Size=7000,Time=2.5606155395507812)(Size=7000,Time=2.5606)
(Size=8000,Time=3.4543330669403076)(Size=8000,Time=3.4543)
(Size=9000,Time=4.004567623138428)(Size=9000,Time=4.0046)
(Size=10000,Time=4.949920415878296)(Size=10000,Time=4.9499)
(Size=50000,Time=121.89765810966492)(Size=50000,Time=121.8977)
```

Using wraps and defining the timer function

9

```python
import time
from functools import wraps


def timer(algo):
    @wraps(algo)
    def test(data,log = False, output = False):
        if log:
            print("Data before %s :%s"%(algo.__name__,str(data)))
        time_stdatart=time.time()
        res = algo(data)
        t = time.time() - time_stdatart
        if log:
            print("Data after %s :%s"%(algo.__name__,str(data)))
            print("Runnint time %s s"%(str(t)))
        return [algo.__name__,t,res] if output else [algo.__name__,t]
    return test
```


Wrapping Timer over an algo

10

```python
@timer ←————————
def SelectionSort(arr):

    # Python program for implementation of Selection
    # Sort


    # Traverse through all array elements
    for i in range(len(arr)):

        # Find the minimum element in remaining
        # unsorted array
        min_idx = i
        for j in range(i+1, len(arr)):
            if arr[min_idx] > arr[j]:
                min_idx = j

        # Swap the found minimum element with
        # the first element
        arr[i], arr[min_idx] = arr[min_idx], arr[i]
```

Timing Selection Sort

11

```
28 #Using Wrapper
29 SelectionSort(data,log=True)
30
```

```
Data before SelectionSort :[24715, 38873, 18596, 22201, 22002, 24400, 45849, 24750, 4067, 56838, 21424, 15479, 47269, 54093,
49421, 54948, 22812, 10012, 36799, 39986]
Data after SelectionSort :[4067, 10012, 15479, 18596, 21424, 22002, 22201, 22812, 24400, 24715, 24750, 36799, 38873, 39986,
45849, 47269, 49421, 54093, 54948, 56838]
Runnint time 0.0 s
```

```
31 #Functools
32 algos=[SelectionSort]
33 for algo in algos:
34     algo(data, log=True)
35
```

```
Data before SelectionSort :[37012, 48401, 23941, 65328, 19322, 35656, 17745, 48773, 39621, 15411, 39202, 237, 37730, 60833,
20957, 57983, 64392, 28430, 37706, 64305]
Data after SelectionSort :[237, 15411, 17745, 19322, 20957, 23941, 28430, 35656, 37012, 37706, 37730, 39202, 39621, 48401, 4
8773, 57983, 60833, 64305, 64392, 65328]
Runnint time 0.0 s
```


Four Distinct Areas of Study

12

- How to devise algorithms
  - Selection Sort method, what other methods?
- How to validate algorithms
  - How to prove correctness?
- How to analyze algorithms
  - What about performance? Theoretically and in practice?
- How to test algorithms
  - Correctness
  - Performance

Devise, Validate, Analyze, and Test

13

```
# Python program for implementation of Selection
# Sort


# Traverse through all array elements
for i in range(len(arr)):

    # Find the minimum element in remaining
    # unsorted array
    min_idx = i
    for j in range(i+1, len(arr)):
        if arr[min_idx] > arr[j]:
            min_idx = j

    # Swap the found minimum element with
    # the first element
    arr[i], arr[min_idx] = arr[min_idx], arr[i]
```

```
#Number of steps:
    #min_idx - n steps

    #in the j loops
    # at most 2(n-1) + 2(n-2) + ... + 2(1)
    # = 2((n-1)+1)*(n-1))/2
    # = n(n-1)
    # = n^2 + n

    #swap
    #2n

    #Total = n + n^2 +2n
    #      = 3n + n^2
```

13

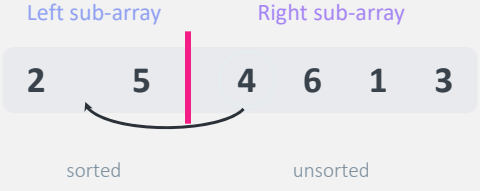
Insertion Sort

14

Input array

5 2 4 6 1 3

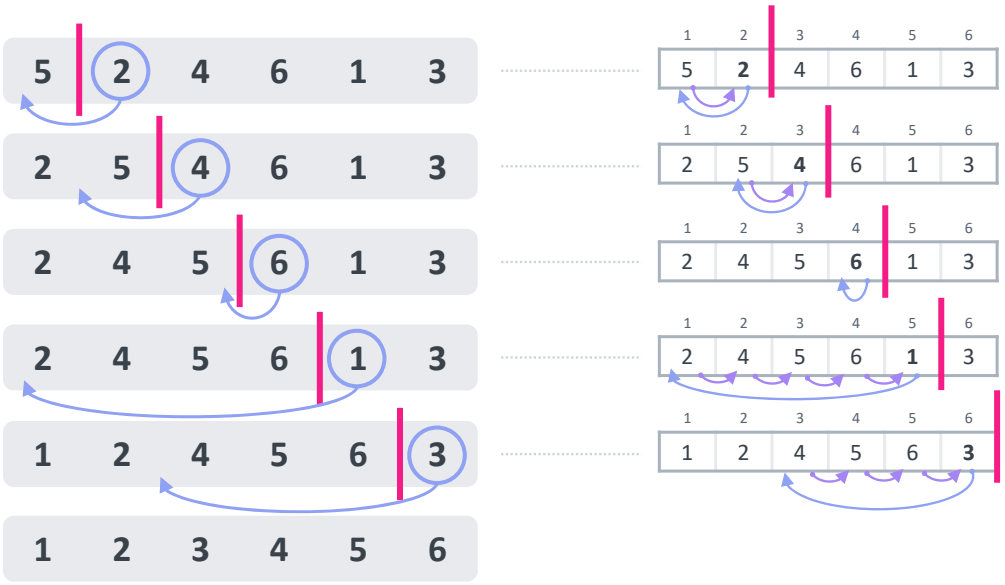
at each iteration, the array is divided in two sub-arrays:



14

Insertion Sort

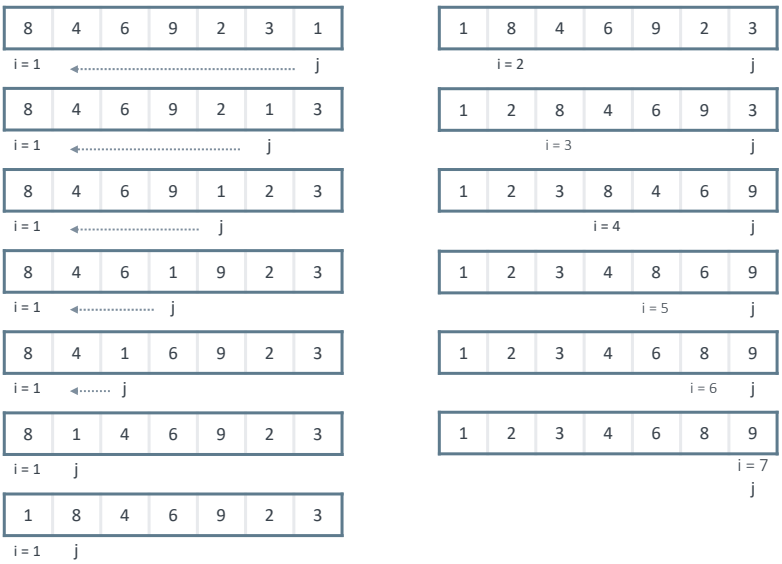
15



15

Bubble Sort

16



16

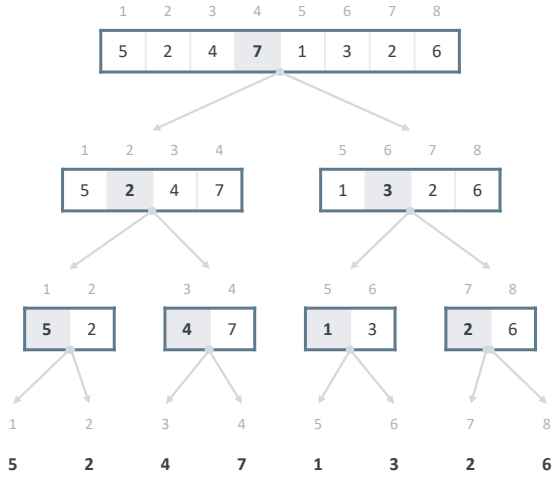
The Divide and Conquer Method

17

```
DC(P):
 if small(P):
    return(P)
 else:
    divide P into smaller instances
    P1, P2, … Pk, K>=1
    apply divide and conquer to each of these
    sub-problems
    return combine(DC(P1), DC(P2), … , DC(Pn))
```

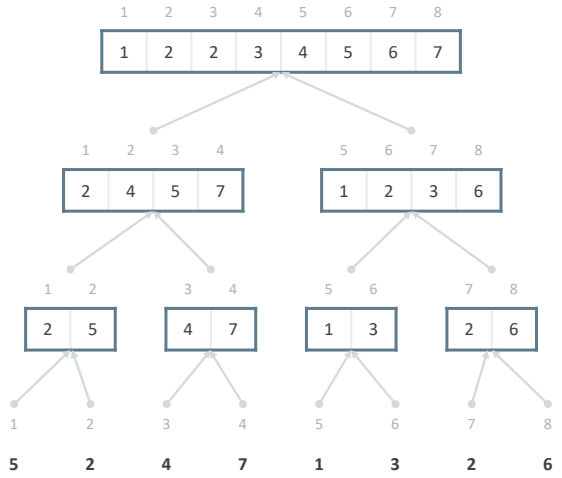

Mergesort Divide

18



Mergesort Conquer and Merge

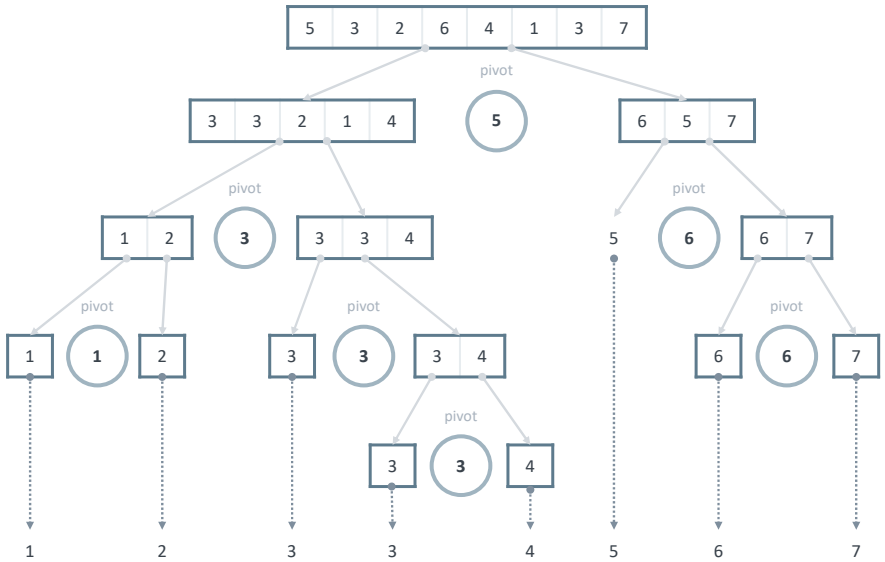
19



19

Quicksort

20



20

Type of Analysis

21

- Worst case
  - Provides an upper bound on running time
- Best case
  - Provides a lower bound on running time
- Average case
  - Provides a prediction about the running time
  - Assumes that the input is random
- ** Benchmark case
  - Provide the prediction about the running on cases that are relevant to the problem the algorithm is solving

What to compare objectively?

22

- Compare execution times?
- Count the number of statements executed?
- Express running time as a function of the input size *n* (i.e., *f(n)*).
  - Compare different functions corresponding to running times.
  - Such an analysis is independent of machine time, programming style, etc.

Asymptotic Analysis

23

- To compare two algorithms with running times *f(n)* and *g(n)*, we need a rough measure that characterizes how fast each function grows.
- We use rate of growth
- Compare functions in the limit, that is, asymptotically! i.e., for large values of *n*

Rate of Growth

24

- The low order terms in a function are relatively insignificant for large n

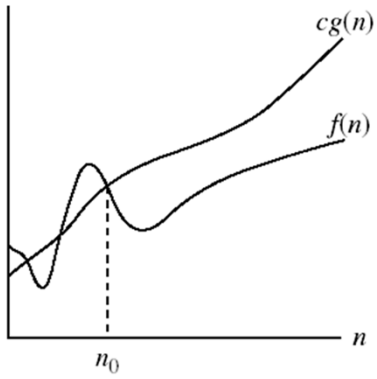
$$n^4 + 10n^3 + 100n^2 + 1000n + 10 \approx n^4$$

- We say that $n^4 + 10n^3 + 100n^2 + 1000n + 10$ and n^4 have the same rate of growth

Asymptotic Notation

25

$O(g(n)) = \{f(n) :$ there exist positive constants c and n_0 such that
$0 \le f(n) \le cg(n)$ for all $n \ge n_0\}$.

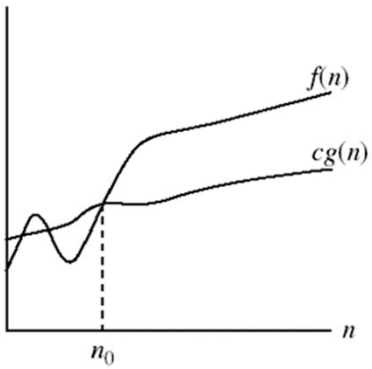


$g(n)$ is an ***asymptotic upper bound*** for $f(n)$.

Asymptotic Notation

26

$\Omega(g(n)) = \{f(n) :$ there exist positive constants c and n_0 such that
$0 \le cg(n) \le f(n)$ for all $n \ge n_0\}$.

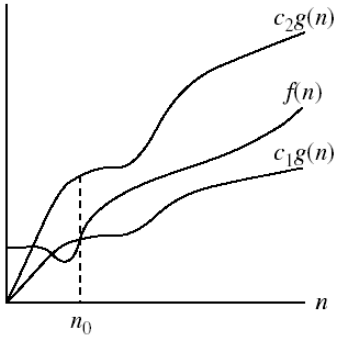


$g(n)$ is an ***asymptotic lower bound*** for $f(n)$.

Asymptotic Notation

27

$\Theta(g(n)) = \{f(n)$: there exist positive constants c_1, c_2, and n_0 such that $0 \le c_1 g(n) \le f(n) \le c_2 g(n)$ for all $n \ge n_0\}$.



$\Theta(g(n))$ is the set of functions with the same order of growth as g(n)

$g(n)$ is an ***asymptotically tight bound*** for $f(n)$.

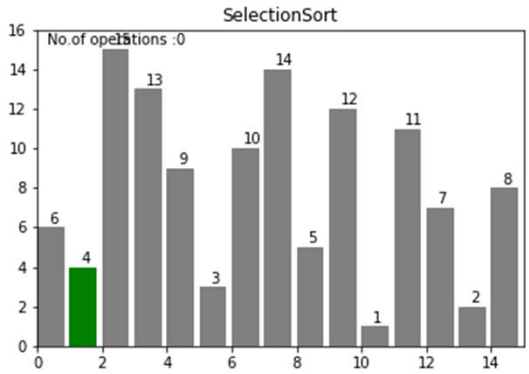
O-Notation

28

- $n^4 + 10n^3 + 100n^2 + 1000n + 10$ is $O(n^4)$
- 12345 is O(1)
- Selection Sort?
- Bubblesort?
- Insertion sort?
- Mergesort?
- Quicksort?
- Best Case? And Average case?

Sorting Visualization – Selection and Insertion Sorts

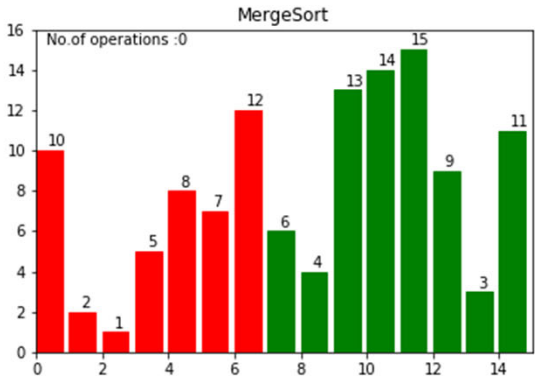
29

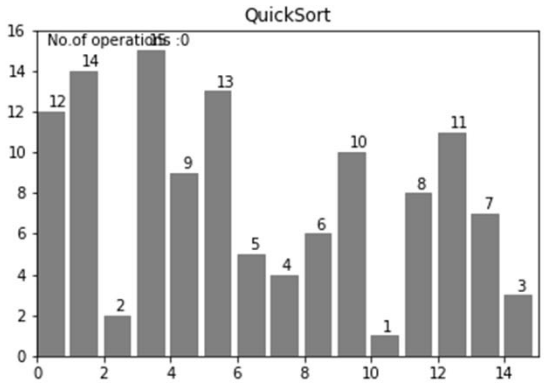


29

Sorting Visualization – Merge and Quick Sorts

30

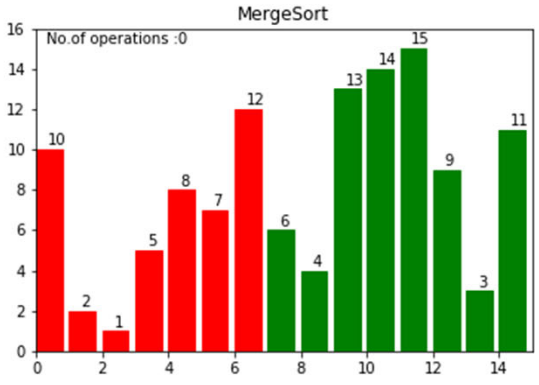


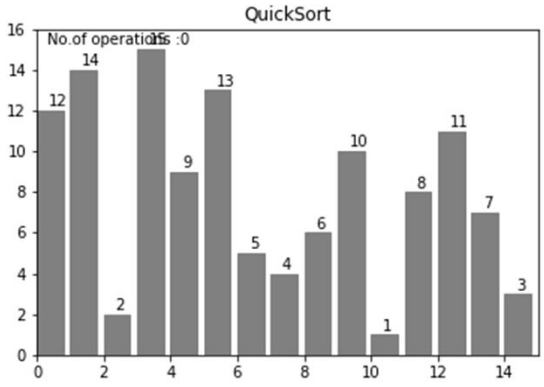


30

Mind Numbing Sorting Visualization

31





The Beauty of Fibonacci

32

Ways and Implications to code the Fibonacci function

33

```
def Fib(n):
  if (n<2):
    return n
  return (Fib(n-1))+(Fib(n-2))
```

```
def Fib1(n):
    global f
    if (f[n]<0):
        temp=Fib1(n-1)+Fib1(n-2)
        f[n]=temp
    return f[n]
```


Undoing the Recursion

34

```
def Fib_loop(n):
    res = 0
    a1 = 1
    a2 = 1
    for i in range(n-2):
        res = a1 + a2
        a1 = a2
        a2 = res
    return res
```

Relationship between Fibonacci and Golden Ratio

35

$$f_n = \frac{1}{\sqrt{5}}\left(\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n\right)$$

Beautiful Mathematics

```
import math
def Fib_clos(n):
    phi1 = (1+math.sqrt(5))/2
    phi2 = (1-math.sqrt(5))/2
    return (math.pow(phi1,n)-math.pow(phi2,n))/math.sqrt(5)
```


Fast Exponentiation

36

$$x^n = \begin{cases} x\left(x^2\right)^{\frac{n-1}{2}}, & \text{if } n \text{ is odd} \\ \left(x^2\right)^{\frac{n}{2}}, & \text{if } n \text{ is even.} \end{cases}$$

Applying it to Fibonacci

37

$$\begin{pmatrix} F_1 \\ F_2 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} * \begin{pmatrix} F_0 \\ F_1 \end{pmatrix}$$

And similarly:

$$\begin{pmatrix} F_2 \\ F_3 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} * \begin{pmatrix} F_1 \\ F_2 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^2 * \begin{pmatrix} F_0 \\ F_1 \end{pmatrix}$$

Which in general turns into:

$$\begin{pmatrix} F_n \\ F_{n+1} \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^n * \begin{pmatrix} F_0 \\ F_1 \end{pmatrix}$$

We know that F_0 and F_1 = 0, 1 so we can put the whole thing into python and see how it fairs compared to our polynomial algorithm

Identifying the Repeated Element

38

- Consider an array of a[] of n numbers that has n/2 distinct elements and n/2 copies of another element. Propose an algorithm to find the repeated element
- Method 1?

- Method 2?

Identifying the Repeated Element

39

```
import timeit

def simple_repeat(a):
    n = len(a)
    for i in range(n):
        for j in range(i+1,n):
            if(a[i]==a[j]):
                return a[i]
    return -1
```

```
def sortandcheck(a):
    a.sort()
    n = len(a)
    for i in range(n-1):
        if (a[i]==a[i+1]):
            return a[i]
    return -1
```

```
def rand_repeat(a):
    n = len(a)
    while True:
        i = random.randint(0,n-1)
        j = random.randint(0,n-1)
        if a[i]==a[j] and i!=j:
            return a[i]
```


Identifying the Repeated Element – The Result

40

```
1 n = 5000*2
2 arr = [i for i in range(n//2)] + [n]*(n//2)
3
4 print("simple algo %f s"%(timeit.timeit("simple_repeat(arr)","from __main__ import arr,simple_repeat",number=50)))
5 print("sortandcheck algo %f s"%(timeit.timeit("sortandcheck(arr)","from __main__ import arr,sortandcheck",number=50)))
6 print("randomized algo %f s"%(timeit.timeit("rand_repeat(arr)","from __main__ import arr,rand_repeat",number=50)))
```

```
simple algo 129.101156 s
sortandcheck algo 0.028360 s
randomized algo 0.000497 s
```

Identifying the Repeated Element

41

- What is the probability that in an iteration repeated elements are found and iteration will quit is:

$$\frac{\frac{n}{2}\left(\frac{n}{2}-1\right)}{n^2} \geq \frac{1}{5} \quad \text{for all} \quad n \geq 10$$

- This means that it won't quit $< \frac{4}{5} = 0.8$
- Probability that it won't quit in 10 iterations is $< 0.8^{10} = 0.1073741824$
- 100 iterations is 2.03074e-10
- 200 iterations is 4.14951e-20

Divide and Conquer Method

42

```
DC(P):
 if small(P):
    return(P)
 else:
    divide P into smaller instances
    P1, P2, … Pk, K>=1
    apply divide and conquer to each of these
    sub-problems
    return combine(DC(P1), DC(P2), … , DC(Pn))
```

Binary Search


```
1  def search(arr, l, r, key):
2      if (l > r): return -1
3      
4      mid = (l + r) // 2
5      if (arr[mid] > key):
6          return search(arr, l, mid - 1, key)
7      elif (arr[mid] < key):
8          return search(arr, mid + 1, r, key)
9      else:
10         return mid
11 a = [1,2,3,4,5,6,7,8,9,10]
12 search([1,2,3,4,5,6,7,8,9,10], 0, len(a) - 1,1)
```

```
0
```


Divide and Conquer – Integer Multiplication


```
1234 x 1234 = ?

   1234
  x1234
-------
   4936
  37020
 246800
1234000
-------
1522756
```

1234×1234. Let $a = 12, b = 34$

Step		
1	ac	$= 12 \text{ x } 12 = 144$
2	bd	$= 34 \text{ x } 34 = 1156$
3	$(a+b) \times (c+d)$	$= (12+34)\text{x}(12+34)$
		$= 46 \text{ x } 46 = 2116$
4	(3) - (2) - (1)	$= 2116\text{-}1156\text{-}144$
5	(1)x10000+(2)+(4)x100	$= 1522756$

Shifting of some partial results in Step (5) is not considered as multiplication. As a result,
$T(n) = 3T(\frac{n}{2}) + cn \Rightarrow T(n) = n^{\log_2 3} = n^{1.585}$

Divide and Conquer – Find Min-Max in a list?

45

```
def minmax1(arr):
    min=a[0]
    max=a[0]
    for i in range(1,len(a)):
        if (a[i]<min): min=a[i]
        if (a[i]>max): max=a[i]
    return(min,max)
```

```
def minmax1(arr):
    min=a[0]
    max=a[0]
    for i in range(1,len(a)):
        if (a[i]<min): min=a[i]
        else:
            if (a[i]>max): max=a[i]
    return(min,max)
```


Divide and Conquer – Find min and max

46

```
import random

def minandmax(arr, l, r):
    if r - l + 1 <= 2:
        return (min(arr[l],arr[r]), max(arr[l],arr[r]))
    mid = (l+r) // 2
    arr1min, arr1max = minandmax(arr, l, mid)
    arr2min, arr2max = minandmax(arr, mid+1, r)
    return (min(arr1min, arr2min), max(arr1max, arr2max))

a = [62, 94, 11, 91, 90, 15, 40, 53, 56, 70, 55, 5, 20, 10, 8, 37, 19, 50, 93, 10]
print(a)
print(minmax1(a))
print(minmax2(a))
print(minandmax(a, 0, a.__len__() - 1))
print(dir(a))
```

```
[62, 94, 11, 91, 90, 15, 40, 53, 56, 70, 55, 5, 20, 10, 8, 37, 19, 50, 93, 10]
(5, 94)
(5, 94)
(5, 94)
['__add__', '__class__', '__contains__', '__delattr__', '__delitem__', '__dir__', '__doc__', '__eq__', '__format__', '__ge_
_', '__getattribute__', '__getitem__', '__gt__', '__hash__', '__iadd__', '__imul__', '__init__', '__init_subclass__', '__ite
r__', '__le__', '__len__', '__lt__', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__reversed_
_', '__rmul__', '__setattr__', '__setitem__', '__sizeof__', '__str__', '__subclasshook__', 'append', 'clear', 'copy', 'coun
t', 'extend', 'index', 'insert', 'pop', 'remove', 'reverse', 'sort']
```

Divide and Conquer – Find Min-Max in a list?

47

Assuming that $n = 2^k$

$$
\begin{aligned}
T(n) &= 2T(\tfrac{n}{2}) + 2 \\
&= 2(2T(\tfrac{n}{4}) + 2) + 2 \\
&= 4T(\tfrac{n}{4}) + 4 + 2 \\
&\cdots \\
&= 2^{k-1}T(2) + \sum_{i=1}^{k-1} 2^i \\
&= 2^{k-1} + 2^k - 2 \\
&= \tfrac{n}{2} + n - 2 = \tfrac{3n}{2} - 2
\end{aligned}
$$

is the best, average and worst case.

Divide and Conquer – Matrix Multiplication

48

$$
\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}
$$

$$
\begin{aligned}
C_{11} &= A_{11} \times B_{11} + A_{12} \times B_{21} \\
C_{12} &= A_{11} \times B_{12} + A_{12} \times B_{22} \\
C_{21} &= A_{21} \times B_{11} + A_{22} \times B_{21} \\
C_{22} &= A_{21} \times B_{12} + A_{22} \times B_{22}
\end{aligned}
$$

$$
T(n) = 8T(\tfrac{n}{2}) + O(n^2) \Rightarrow T(n) = O(n^3)
$$

Divide and Conquer – Matrix Multiplication

$$
\begin{aligned}
P &= (A_{11} + A_{22})(B_{11} + B_{22}) \\
Q &= (A_{21} + A_{22})B_{11} \\
R &= A_{11}(B_{12} - B_{22}) \\
S &= A_{22}(B_{21} - B_{11}) \\
T &= (A_{11} + A_{12})B_{22} \\
U &= (A_{21} - A_{11})(B_{11} + B_{12}) \\
V &= (A_{12} - A_{22})(B_{21} + B_{22})
\end{aligned}
\qquad
\begin{aligned}
C_{11} &= P + S - T + V \\
C_{12} &= R + T \\
C_{21} &= Q + S \\
C_{22} &= P + R - Q - U
\end{aligned}
$$

$$
T(n) = \begin{cases} b & n \leq 2 \\ 7T(\frac{n}{2}) + an^2 & n > 2 \end{cases}
$$

$$
T(n) = \Theta(n^{\log_2 7}) = \Theta(n^{2.81})
$$

Matrix Multiplication – Python Implementation


```python
import numpy as np

def submat(mat):
    n, _ = mat.shape
    nn = n//2
    return mat[:nn, :nn], mat[:nn, nn:], mat[nn:, :nn], mat[nn:, nn:]

def strassen(a, b):
    if len(a) == 1:
        return a * b
    a11, a12, a21, a22 = submat(a)
    b11, b12, b21, b22 = submat(b)
    m = [None,
         strassen(a11 + a22, b11 + b22),
         strassen(a21 + a22, b11),
         strassen(a11, b12 - b22),
         strassen(a22, b21 - b11),
         strassen(a11 + a12, b22),
         strassen(a21 - a11, b11 + b12),
         strassen(a12 - a22, b21 + b22)]
    c11 = m[1] + m[4] - m[5] + m[7]
    c12 = m[3] + m[5]
    c21 = m[2] + m[4]
    c22 = m[1] - m[2] + m[3] + m[6]
    c = np.vstack((np.hstack((c11, c12)), np.hstack((c21, c22))))
    return c


A = np.random.randint(0,100,size = (8,8))
B = np.random.randint(0,100,size = (8,8))
print(A)
print(B)
print(strassen(A, B))
```

Result of the Multiplication

```
[[45 82 21 76 13 27 16 96]
 [44 24 56 77 79 72 84 98]
 [55 69 18 75 98 43 65 79]
 [43 20 64 43 94 86  2 65]
 [ 8 10 62 66 19 86 28 14]
 [78 53 70 29 37 35 75 78]
 [ 4 89 49 94 15 50  4  9]
 [82 22 24 54 90 66 64 93]]
[[80 97 78 98  2 38  2  8]
 [ 9 38 85 49 54 20 96 96]
 [70 54 79 81 98 45 93 72]
 [84 27 48 43 88 72 38 25]
 [25  5 12 80 68 79 20 98]
 [79 32 37 74 99 57 44 76]
 [ 9 61 20 80 67  7 96 12]
 [ 6 89 50 48 54 22 94 59]]
[[15370 21116 22062 22323 23077 14557 24811 20826]
 [23131 26828 23784 36407 37068 23305 32550 28617]
 [19487 23816 23194 33468 31742 22066 28776 28548]
 [21264 18677 19774 29391 29772 21782 21558 27554]
 [18219 12087 14210 19930 24878 14953 17418 16698]
 [18886 26965 24650 32452 27648 16421 29668 23265]
 [16862 11674 18820 18416 24612 15166 20411 20303]
 [21572 26287 22206 35048 30420 21890 26150 25937]]
```

Interesting Challenge

51

What is the minimum number of comparisons needed to find the smallest and the second smallest items in a list of *n* items?

The Greedy Method -Framework

52

```
The important parts:
1. A candidate set, from which a solution is created
2. A selection function, which chooses the best candidate to be added to the solution
3. A feasibility function, that is used to determine if a candidate
   can be used to contribute to a solution
4. An objective function, which assigns a value to a solution, or a partial solution, and
5. A solution function, which will indicate when we have discovered a complete solution

S is empty
While candidate set is not empty and complete solution not found
    select the best candidate C
    add C to S if possible

if C is a complete solution, solution found
else solution not found
```

The Fractional Knapsack Problem

53

$$\max \sum_{i=1}^{n} p_i x_i$$

such that

$$\sum_{i=1}^{n} w_i x_i \leq m,\ 0 \leq x_i \leq 1 \text{ and } 1 \leq i \leq n$$

$n = 3, m = 20$
$(p_1, p_2, p_3) = (25, 24, 15)$
$(w_1, w_2, w_3) = (18, 15, 10)$

x_1	x_2	x_3	$\sum w_i x_i$	$\sum p_i x_i$
1/2	1/3	1/4	16.5	24.25
1	2/15	0	20	28.2
0	2/3	1	20	31
0	1	1/2	20	31.5

Proof of Correctness

54

General Principle of the proof:

1. Compare the greedy solution with any optimal solution.
2. If the solutions differ, find the first x_i which they differ.
3. Make the x_i in the optimal solution equal to that of the greedy solution with any loss in the total value.
4. Repeat (1) to (3). Hence the greedy solution is also optimal.

Proof of Correctness

55

Given that $\frac{p_1}{w_1} \geq \frac{p_2}{w_2} \geq \ldots \frac{p_n}{w_n}$, our greedy method generates an optimal solution.
Let $X = (x_1, x_2, \ldots, x_n)$, if $x_i = 1, \forall i$, then the solution is optimal. If not, let j be the least index where $x_j \neq 1$.
$x_i = 1, 1 \leq i < j$ and $x_i = 0, j < i \leq n$ and $0 \leq x_j < 1$.
Let $Y = (y_1, y_2, \ldots, y_n)$ be an optimal solution.
Let k be the least index wehre $x_k \neq y_k$. Such a k must exist, else $X = Y$.
It follows that $x_k > y_k$ because if $x_k = 1$ and $y_k \neq x_k$ if $k < j$, this implies that $y_k < x_k$.
If $k = j$, $\sum_{i=1}^{k} w_i x_i = m$, if $y_k > x_k$ this implies that $\sum_{i=1}^{k} w_i y_i > m$. Thus $y_k < x_k$.
If $k > j$, then $\sum_{i=1}^{k} w_i y_i > m$, and this is not possible.

Proof of Correctness - continues

56

Suppose we increase y_k to x_k and decrease as many of $(y_{k+1}, \ldots, y_n)$ as necessary so that the total capacity used is still m. Let the new solution be $Z = (z_1, z_2, \ldots, z_n)$.
$z_i = x_i, \forall i, 1 \leq i \leq k$ and $\sum_{i=k+1}^{n} w_i(y_i - z_i) = w_k(z_k - y_k)$.

$$\sum_{i=1}^{n} p_i z_i = \sum_{i=1}^{n} p_i y_i + (z_k - y_k)p_k - \sum_{k<i\leq n}(y_i - z_i)p_i$$

$$= \sum_{i=1}^{n} p_i y_i + (z_k - y_k)w_k \frac{p_k}{w_k} - \sum_{i=k+1}^{n}(y_i - z_i)w_i \frac{p_i}{w_i}$$

$$\geq \sum_{i=1}^{n} p_i y_i + \frac{p_k}{w_k}[(z_k - y_k)w_k - \sum_{i=k+1}^{n}(y_i - z_i)w_i]$$

This implies $\sum_{i=1}^{n} p_i z_i \geq \sum_{i=1}^{n} p_i y_i$.

Proof of Correctness – the end

57

If $\sum p_i z_i > \sum p_i y_i$ then Y could not be optimal.

If $\sum p_i z_i = \sum p_i y_i$ then either $Z = X$ or $Z \neq X$.

If $Z = X$, then X is optimal. If $Z \neq X$, repeat the use of the same transformation until Y becomes X.

As such X is optimal.

Job Sequencing with Deadlines

58

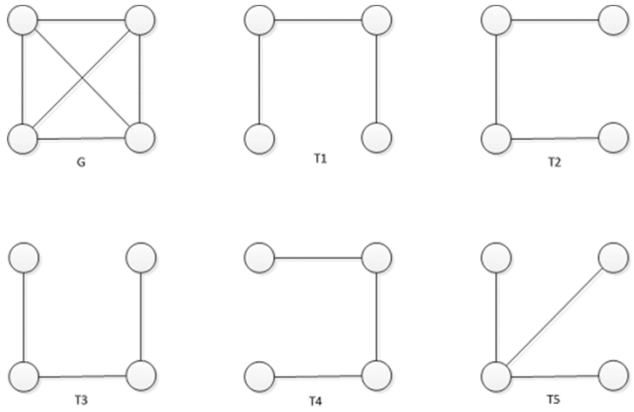
$n = 4$ $P = (p_1, p_2, p_3, p_4) = (100, 10, 15, 27)$
$D = (d_1, d_2, d_3, d_4) = (2, 1, 2, 1)$

	Feasible Solution	*Processing Sequence*	*Profit*
1	(1,2)	(2,1)	110
2	(1,3)	(1,3) or (3,1)	115
3	(1,4)	(4,1)	127
4	(2,3)	(2,3)	25
5	(3,4)	(4,3)	42
6	(1)	(1)	100
7	(2)	(2)	10
8	(3)	(3)	15
9	(4)	(4)	27

Graph, Subgraph, and Spanning Trees

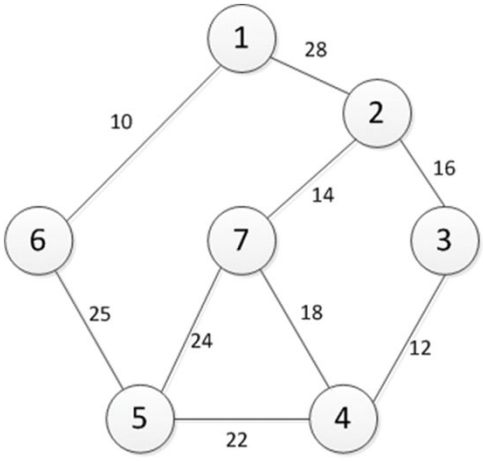
59

Let $G = (V, E)$ be an undirected connected graph. A subgraph $T = (V, E')$ is a spanning tree of G if T is a tree and $E' \subseteq E$. An example is given below where $T1, T2, T3, T4, T5$ are subgraphs of G.

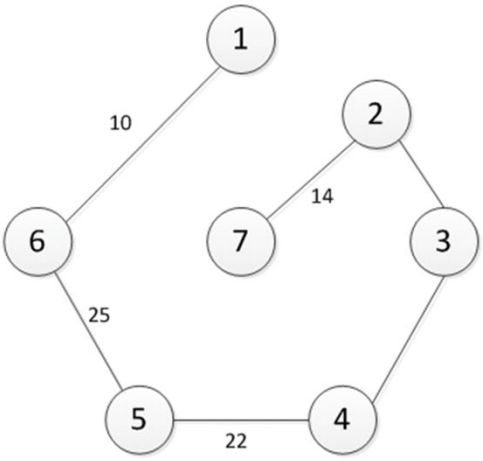


Minimum Cost Spanning Tree

60



Graph G

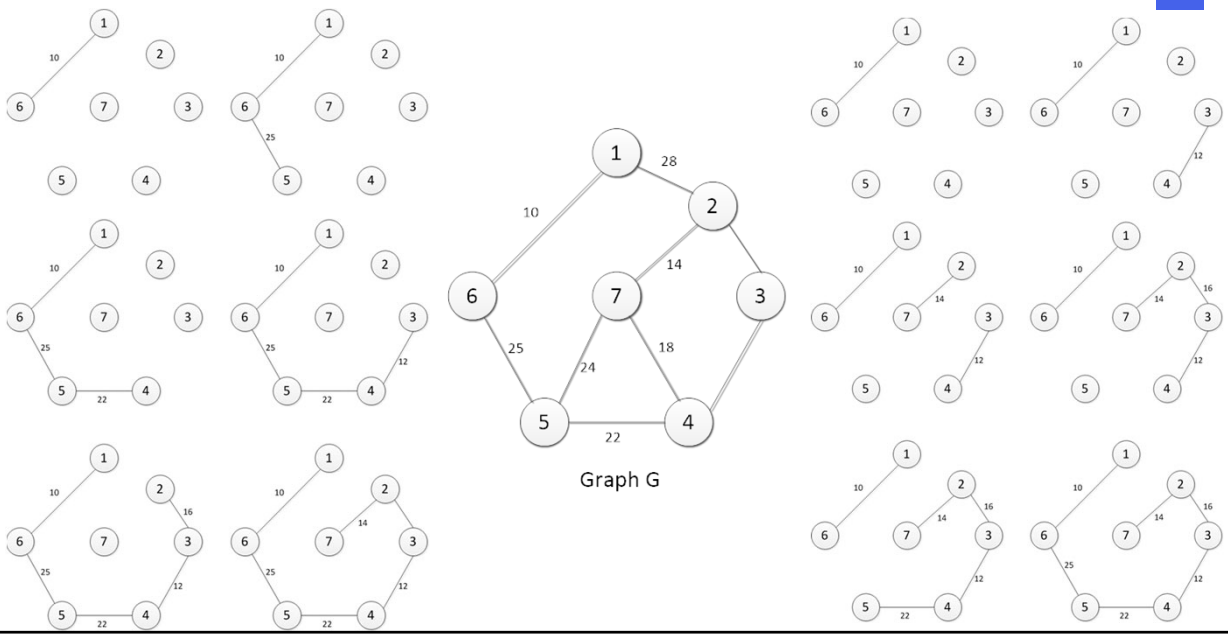


Min cost spanning tree G'

Prim's Method

Kruskal's Method

61

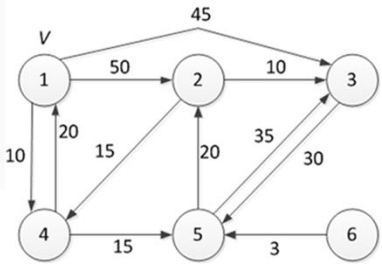


61

Shortest Path

62

```
for each vertex v in Graph:                 // Initialization
    dist[v] := infinity                     // initial distance from source to vertex v is set to infinite
    previous[v] := undefined                // Previous node in optimal path from source
dist[source] := 0                           // Distance from source to source
Q := the set of all nodes in Graph          // all nodes in the graph are unoptimized - thus are in Q
while Q is not empty:                       // main loop
    u := node in Q with smallest dist[ ]
    remove u from Q
    for each neighbor v of u:               // where v has not yet been removed from Q.
        alt := dist[u] + dist_between(u, v)
        if alt < dist[v]
            dist[v] := alt
            previous[v] := u
return previous[ ]
```



62

Shortest Path Algo: ( ) confirmed, [ ] tentative distance from source *v*