Lecture 10

For one search, then we can use the linear search

But for multiple searches(which is what we normally do), we can sort it and then we can use the binary search.

Divide and conquer algorithm:

* split the problem into several sub problems of the same type
* and solve the sub problems independently
* combine the solutions for each sub problem.

**MERGE SORT:**

How hard is it to merge to sorted lists?

We compare the first two elements and whichever is small then is the element in the merged list.

Order of complexity: Ο(n) where n is the sum of the lists.

* divide the list in half
* and keep dividing it until you get a list of size 1
* size 1 lists are always sorted.
* now merge the sub lists.

What is the order of complexity?

n operations of size 1

n/2 operations of size 2

n/4 of size 4 and so on

Order of complexity is: Ο(n log n)

**How to attack a problem with divide and conquer algorithm?**

generalize:

* is there some way to break this problem down to smaller version of the same problem
  + how much division do we have to make

it may be half or it can be different also

* + what is the base case?

when do we get down to a problem whose solution is trivial

* + how do we combine the sub units together?

if we try to make the problem simpler but the combination is very hard then we have got no advantage.

**“BETTER” TECHNIQUE TO SEARCH THEN BINARY SEARCH: HASHING**

ORDER OF complexity of hashing is: CONSTANT

Hashing maps any type of data into integers. This means that the access time is always constant.

But the trade off is that “WE HAVE TRADED SPACE FOR TIME”.

There is no problem in case of integers or characters.

But we cannot make sure that the input to a hash function can give only one output. It can give us multiple outputs then we have to design the code such that it deals with such things.

**Hashes are really hard to create.**

**Tools that we have seen so far:**

* brute force
* successive approximation
* bisection
* divide and conquer

**EXCEPTIONS:**

Distinction between unhandled and handled exception

* try – except block
  + try : try the code
    - if an exception comes up then it goes to the except block
    - else it skips the except block and goes to the further code

if we are taking a file name from the user then we can use, try if the file exists else do something else if the file does not exist.

we can have tags with an except.

* except IOError
* except IndexError
* except NameError

if the error is this type of error then the exception is handled otherwise it is passed up one level until and unless the exception is handled. The code under try is obviously not executed.

**DISTINCTION BETWEEN AN EXCEPTION AND ASSERT:**

the goal of an assert is: My function will give you the type of output you want iff you give me the input that I am expecting.

exception: you can do anything that you want but if anything goes wrong we will handle it for you.