Performance

What is the worst case time complexity of Binary Search?

Binary Search($a[1, \ldots, N]$, k)

```
                                          Cost   Executions
l = 1, r = N + 1                           c1        1
while l < r                                c2        ??
  m = l + (r-l) // 2                       c3        ??
  if (k == a[m])                           c4        ??
    return True                            c5        0
  else if (k < a[m])                       c6        ??
    r = m                                  c7        ??
  else
    l = m + 1                              c8        ??
return False                               c9        1
```


Intuition: loop executes $\log_2 N$ times.

Performance

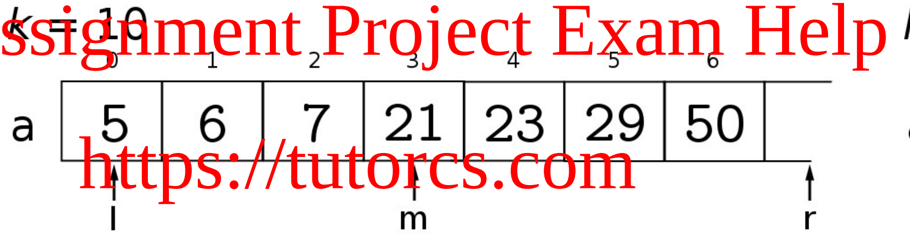
Alternative: analyse the recursive form of the program.

BinSearch(a, l, r, k)


```
                                              Cost
  if (l >= r)                                  c1
    return False                               c2
  m = l + (r-l) / 2                            c3
  if (k == a[m])                               c4
    return True                                c5
  else if (k < a[m])                           c6
    return BinSearch(a, l, m, k)               T(N')
  else
    return BinSearch(a, m+1, r, k)             T(N'')
```

- where N' and N'' are numbers left to search
- Exercise: what are N' and N'' in the worst case? Be exact.

Worst Case Recursion



- `m` is always placed at `l` + $\lfloor N/2 \rfloor$
- if N is odd: $N' = N'' = \lfloor N/2 \rfloor$
- if N is even: $N' = \lfloor N/2 \rfloor$, $N'' = \lfloor N/2 \rfloor - 1$
- So the worst case is when $k < a[0]$
  - If $N > 0$, will have $\lfloor N/2 \rfloor$ unsearched elements

Performance

We can now write a recursive worst case formula for $T(N)$

BinSearch(a, l, r, k)

```
                                                Cost
if (l >= r)                                     c1
  return False                                  c2
m = l + (r-l) / 2                               c3
if (k == a[m])                                  c4
  return True                                   c5
else if (k < a[m])                              c6
  return BinSearch(a, l, m, k)                  T(floor(N/2))
else
  return BinSearch(a, m+1, r, k)                <= T(floor(N/2))
```

Divide and Conquer

Binary Search is a divide and conquer algorithm

- The overall problem is divided into smaller subproblems
- Subproblems must be solved
- The solutions may need to be combined

General form of time complexity is expressed recursively:

$$T(N) = \begin{cases} \Theta(1) & , N < c \\ aT(N/b) + D(N) + C(N) & , \text{otherwise} \end{cases}$$

where c is some small positive integer, a is number of subproblems, N/b is size of a subproblem, $D(N)$ is cost of division and $C(N)$ is cost of combination.

- The "otherwise" formula is a recurrence

Performance

Question

What are a, b, c, $D(N)$ and $C(N)$ for Binary Search?

BinSearch(a, l, r, k)


```
                                          Cost
if (l >= r)                               c1
  return False                            c2
m = l + (r-l) / 2                         c3
if (k == a[m])                            c4
  return True                             c5
else if (k < a[m])                        c6
  return BinSearch(a, l, m, k)            T(floor(N/2))
else
  return BinSearch(a, m+1, r, k)          <= T(floor(N/2))
```

Performance

For Binary Search we have

$$T(N) = \begin{cases} c_1 + c_2 & \text{, if } N = 0 \\ c_1 + c_3 + c_4 + c_6 + T(\lfloor N/2 \rfloor) & \text{, if } N > 0 \end{cases}$$

or

$$T(N) = \begin{cases} \Theta(1) & \text{, if } N = 0 \\ T(\lfloor N/2 \rfloor) + \Theta(1) & \text{, if } N > 0 \end{cases}$$

- Still need to solve the recurrence
- Either: guess answer and prove by induction (beyond this course)
- Or: apply the master method

The Master Method

The outcome of the master method is determined by which of

- the work to solve the base cases: $\Theta(N^{\log_b a})$
- the work to divide and recombine at the top level: $\Theta(f(N))$
- (note $N^{\log_b a}$ is how many base cases, each one is $\Theta(1)$)

is (strictly) polynomially larger.

- If the base case work is larger then $T(N) = \Theta(N^{\log_b a})$
- If neither is larger, then $T(N) = \Theta(N^{\log_b a} \log_2^{k+1} N)$
- If the divide and combine work is larger, then $T(N) = \Theta(f(N))$

Look Out!!!

Polynomially larger is not the same as asymptotically larger. So $N \log_2 N \neq \Omega(N^c)$ for any $c > 1$.

The Master Method [Bentley, Haken, Saxe 1980]

Theorem (Master theorem)

Let a and b be positive real numbers with $a \geq 1$ and $b > 1$. Let $f(N)$ be a function and let $T(N)$ be defined on the non-negative integers by the recurrence:

$$T(N) = aT(N/b) + f(N)$$

where N/b can be replaced by either $\lfloor N/b \rfloor$ or $\lceil N/b \rceil$. Then $T(N)$ has the following asymptotic bounds:

1. *If $f(N) = O(N^c)$ and $c < \log_b a$, then $T(N) = \Theta(N^{\log_b a})$*
2. *If $f(N) = \Theta(N^{\log_b a} \log_2^k N)$ then $T(N) = \Theta(N^{\log_b a} \log_2^{k+1} N)$ for $k \geq 0$*
3. *If $f(N) = \Omega(N^c)$, and $c > \log_b a$, and $af(N/b) \leq cf(N)$ for some $c < 1$ and all sufficiently large N, then $T(N) = \Theta(f(N))$.*

Performance

For Binary Search (worst case)

- $T(N) = T(\lfloor N/2 \rfloor) + \Theta(1)$

So, $N^{\log_b a} = N^{\log_2 1} = N^0 = 1$, and therefore

- $f(N) = \Theta(N^{\log_b a})$

and Case 2, with $k = 0$, applies.

- $T(N) = \Theta(\log_2 N)$

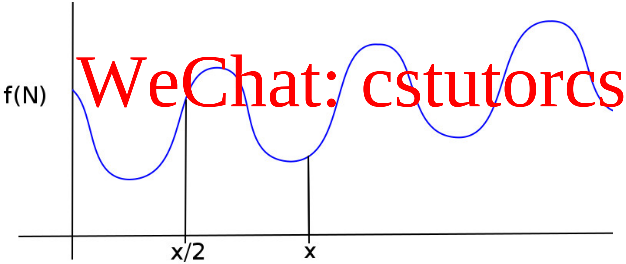
The master method confirms the informal result.

Master Method Case 3

The conditions for Case 3 include an extra check:

- $af(N/b) \leq cf(N)$ where $c < 1$ for all $N > N_0$

This is the so-called regularity condition. It confirms that the divide and combine work decreases as the recursion proceeds. If this is not true the the master theorem does not apply. e.g. for this $f(N)$ there is no N beyond which $f(N) > f(N/2)$



Other Excluded Cases

The master method does not apply to these recurrences

- $T(N) = NT(N/2) + N$
- $T(N) = 0.5T(N/2) + 4N$
- $T(N) = 4T(N/8) - N^2$
- $T(N) = 2T(N/2) + N/\log N$

The (mostly straightfoward) reasons are

- the number of subproblems in the first two
- negative divide and combine time (third example)
- negative value of k (fourth example)