PROOF TECHNIQUES

• Proof by construction

• Proof by induction

• Proof by contradiction

asymptotic performance --How does the algorithm behave as the problem -- size gets very large? -- Running time -- Memory / storage requirements – Bandwidth / power requirements / logic gates / etc.

Internal Sort - The data to be sorted is all stored in the computer’s main memory.

External Sort - Some of the data to be sorted might be stored in some external, slower, device.

In Place Sort - The amount of extra space required to sort the data is constant with the input size.

Stability- Elements with equal keys in the same relative order.

Text

Description automatically generated

Insertion is: (big O)O(n^2),(lower bounded) Ω(n)

Text

Description automatically generated

Text, letter

Description automatically generated

Bubble sort

– Repeatedly pass through the array

– Swaps adjacent elements that are out of order

Text

Description automatically generated

Selection sort:

– Find the smallest element in the array

– Exchange it with the element in the first position

– Find the second smallest element and exchange it with the element in the second position

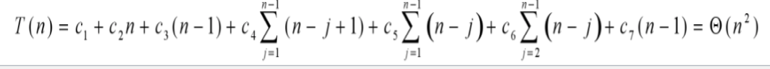
– Continue until the array is sorted

• Disadvantage:

– Running time depends only slightly on the amount of order in the file

Table

Description automatically generated



Text, letter

Description automatically generated

Divide and conquer(merge sort):

- Divide the n-element sequence to be sorted into two subsequences of n/2 elements each

- Sort the two subsequences recursively using merge sort.

- Merge the two sorted subsequences to produce the sorted answer.

|  |  |
| --- | --- |
| ***MergeSort* (*A*, *p*, *r*) //** sort *A*[*p..r*] by divide & conquer   1. **if***p* < *r* 2. **then** *q* ← ⎣(*p*+*r*)/2⎦ 3. *MergeSort* (*A*, *p*, *q*) 4. *MergeSort* (*A*, *q*+1, *r*) 5. *Merge* (*A*, *p*, *q*, *r*) // merges *A*[*p..q*] with *A*[*q+1..r*] | **Merge(*A*, *p*, *q*, *r*)**  1 *n*1 ← *q* – *p* + 1  2 *n*2 ← *r* – *q*   1. **for** *i* ← 1 **to** *n*1 2. **do** *L*[*i*] ← *A*[*p* + *i* – 1] 3. **for** *j* ← 1 **to** *n*2 4. **do** *R*[*j*] ← *A*[*q* + *j*] 5. *L*[*n1*+1] ← ∞ 6. *R*[*n2*+1] ← ∞ 7. *i* ← 1 8. *j* ← 1 9. **for** *k* ←*p* **to** *r* 10. **do if** *L*[*i*] ≤ *R*[*j*] 11. **then** *A*[*k*] ← *L*[*i*] 12. *i* ← *i* + 1 13. **else** *A*[*k*] ← *R*[*j*] 14. *j* ← *j* + 1 |

Text

Description automatically generated

|  |  |
| --- | --- |
| displaymath428 | L’Hopital Rule  Lim f(n) if -> ∞  n->∞ g(n) ∞  then take the derivative of f and g independently  n! is n(n+1)/2  also equal to |

Solving recurrence

Convert the recurrence into a summation and

try to bound it using known series

n Iterate the recurrence until the initial condition is

reached.

n Use back-substitution to express the recurrence in

terms of n and the initial (boundary) condition.

Add:

* Dynamic programming info
* Rod Cutting basics
* Matrix chain multiplication, and info
* Binary search tree info, complexity, algo
* Binary vs binary search
* Balanced tree?
* Quicksort complexities