Algorithm Analysis
Computer Science, UAM

Problems

- **(E)**: exercise; **(E+)**: advanced exercise.
- **(P)**: problem; **(P+)**: advanced problem.
- **(R)**: recommended; it will be solved at the classroom.

1 Abstract runtime of basic algorithms

Exercises

1. **(E; R)** Being `MMul` the traditional algorithm for square matrix multiplication, define a basic operation for this algorithm and estimate $n_{\texttt{MMul}}(A, B)$ for a pair of matrices A, B with dimension N.

2. **(E; R)** The following pseudocode corresponds to an algorithm that finds the minimum number in an array:

```
index Min(array T, index F, index L)
  Min=T[F];
  iMin=F;
  for i=F+1 to L:
    if T[i] < Min:
       Min = T[i];
       iMin=i;
  return iMin;
```

   Analyze its associated runtime using the key comparison as the basic operation.

3. **(E)** Analyze the runtime of the following pseudocode that implements a trivial potentiation algorithm:

```
pot(int x, int N)
  p = x;
  for i=2 to N:
    p = p*x;
  return p;
```

4. **(E; R)** Establish an adequate basic operation and estimate the runtime as a function of n of the following code fragments:

   -
```
sum = 0;
for (i=1; i<=n; i++)
  sum++;
```

-
  ```
  sum = 0;
  for (i=1; i<=n; i++)
    for (j=0; j<n; j++)
      sum++;
  ```

-
  ```
  sum = 0;
  for (i=1; i<=n; i++)
    for (j=0; j<n*n; j++)
      sum++;
  ```

Problems

5. **(P; R)** Establish an adequate basic operation and estimate the runtime as a function of n of the following code fragments:

   -
     ```
     sum = 0;
     for (i=1; i<=n; i++)
       for (j=0; j<i; j++)
         sum++;
     ```

   -
     ```
     sum = 0;
     for (i=1; i<=n; i++)
       for (j=0; j<i*i; j++)
         for (k=0; k<j; k++)
           sum++;
     ```

   -
     ```
     sum = 0;
     for (i=1; i<=n; i++)
       for (j=0; j<i*i; j++)
         if (j%i == 0)
           for (k=0; k<j; k++)
             sum++;
     ```

6. **(P)** Justify the following inequalities $(A + B)/2 \leq \max(A, B) \leq A + B$. What do they imply for the calculation of the number of basic operations on selections?

2 Function growth

Exercises

7. **(E; R)** For a certain problem, we consider five algorithms whose running time in microseconds for an input size N are: $10.000N$, $1.000N \log_{10} N$, $100N^2$, $10N^3$ y $10^{-3}10^{N/10}$, respectively. Build a table withe their running time for the following N values: 10, 100, 1.000, 10.000, 100.000 y 1.000.000.

   What is the maximum input size so that the running time for each algorithm is 1 minute? Perform the same analysis for 1 day running time.

8. **(E; R)** Sort the following functions according to their growth speed: n, $\sqrt{n}$, $n^{1.5}$, n^2, $n \log n$, $n \log\log n$, $n \log^2 n$, $2/n$, 2^n, $2^{n/2}$, 37, $n^2 \log n$, n^3.

9. **(E)** The running times for three algorithms that solve the same problem for input size N are: N^2, $10N(\log_2 N)^2$ y $\frac{1}{8}N^2 \log_2 N$, respectively. Indicate the N range in which each algorithm is more convenient. (Consider power of two numbers for the range limits.)

10. **(E; R)** The analysis of two algorithms, A and B, shows that $W_A(N) = \mathrm{O}(1000N \log_{10}(N))$ and $W_B(N) = \mathrm{O}(N^2)$. What is the best algorithm for "small" problems? What is best for "large" problems? Indicate what would be consider small and large in this context.

11. **(E; R)** Prove that for all $a > 0$, $\log n = \mathrm{o}(n^a)$.

12. **(E+)** Prove that $\lfloor N \rfloor = \Theta(N)$ an also that $\lceil N \rceil = \Theta(N)$ (is it possible to do it using limits?).

13. **(E+)** What is the theoretical growth order of function $f(N) = N \log N + 100N$? What is its "real" growth order?

14. **(E; R)** If $T_1 \approx f$ and $T_2 \approx f$, decide with arguments whether the following statements are true or false (provide a brief argument for the true statements and a counterexample or a counterargument for the false ones):

    (a) $T_1 + T_2 \approx f$;

    (b) $T_1 - T_2 = \mathrm{o}(f)$;

    (c) $T_1 \cdot T_2 \approx f^2$.

15. **(E; R)** If $T_1 \approx f$ and $T_2 \approx f$, decide with arguments whether the following statements are true or false (provide a brief argument for the true statements and a counterexample or a counterargument for the false ones):

    (a) $T_1 + T_2 \approx 2f$;

    (b) $T_1^2 - T_2^2 = \mathrm{o}(f)$;

    (c) $T_1/T_2 \approx 1$.

Problems

16. **(P; R)** If $T_1(N) = \mathrm{O}(f(N))$ and $T_2(N) = \mathrm{O}(f(N))$, which of the following statements are true and which ones are false?

    (a) $T_1(N) + T_2(N) = \mathrm{O}(f(N))$;

    (b) $T_1(N) - T_2(N) = \mathrm{o}(f(N))$;

    (c) $\frac{T_1(N)}{T_2(N)} = \mathrm{O}(1)$;

    (d) $T_1(N) = \mathrm{O}(T_2(N))$.

    (provide a brief argument for the true statements and a counterexample for the false ones).

17. **(P; R)** If $T_1(N) = \Theta(f(N))$ and $T_2(N) = \Theta(f(N))$, which of the following statements are true and which ones are false?

(a) $T_1(N) + T_2(N) = \Theta(f(N))$;

(b) $T_1(N) - T_2(N) = \mathrm{o}(f(N))$;

(c) $\frac{T_1(N)}{T_2(N)} = \Theta(1)$;

(d) $T_1(N) = \Theta(T_2(N))$.

(provide a brief argument for the true statements and a counterexample for the false ones).

18. **(P)** We say that $F = f + \mathrm{O}(g)$ if $g = \mathrm{o}(f)$ and $|F - f| = \mathrm{O}(g)$. Check that $\sum_1^N i^k = N^{k+1}/(K+1) + O(N^K)$.

19. **(P+)** Prove that for all N, $\log N! = \Theta(N \log N)$ y $N! = \mathrm{o}(N^N)$.

20. **(P; R)** Is it true that for any f and for any constant C, $f(Cn) = \mathrm{O}(f(n))$? Is it true that $f(n+C) = \mathrm{O}(f(n))$?

3 Worst, best and average cases for basic algorithms

Exercises

21. **(E; R)** Estimate (with reasoned arguments) the growth of the following function:

$$S(N) = \sum_1^N \frac{1}{i^{1/3}}.$$

22. **(E)** We know for a certain array of size N that $P(T[i]) = \frac{i^2}{C_N}$, where C_N is a normalization constant. What is the average number of key comparisons that the linear search algorithm performs on successful searches?

23. **(E; R)** We would like to use the following pseudo-code for a linear search on sorted arrays:

```
int LSearch (array T, dim N, key K)
  i=1;
  while i <= N and T[i] < K:
    i++;
  if i > N:
    return error;
  else if T[i] != K:
    return error;
  else:
    return i;
```

Assuming that $P(K = i) = P(i < K < i+1) = P(i < 1) = P(i > N) = \frac{1}{2N+1}$, calculate $A_{LSearch}(N)$ considering both successful and unsuccessful searches.

24. **(E; R)** For a given array T, we know that

$$P(K = T[i]) = \frac{1}{C_N}\frac{\log i}{i},$$

where $C_N = \sum_1^N \frac{\log i}{i}$.
Calculate (with reasoned arguments) $A^s_{LSearch}(N)$.

25. **(E)** For a given array T, we know that

$$P(K = T[i]) = \frac{1}{C_N} i \log i,$$

where $C_N = \sum_1^N i \log i$. Calculate (with reasoned arguments) $A^s_{LSearch}(N)$.

Problems

26. **(E)** Prove using induction that $\sum_1^N a + ib = N(a + \frac{N+1}{2}b)$.

27. **(P; R)** Prove using induction that:
$\sum_1^N n^2 = n(n+1)(2n+1)/6$,
$\sum_1^N n^3 = (n(n+1)/2)^2$.

28. **(P+)** Provide an expression for $\sum_1^N nx^n$, and also for $\sum_1^N n^2x^n$. (Note that the former looks like a derivative and the latter is also related).

4 Evolution of local search algorithms

Exercises

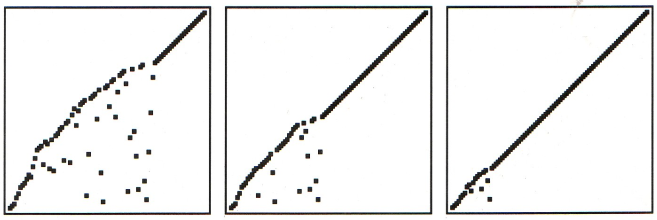
29. **(E)** Given the following list: `[20, 3, 10, 6, 8, 2, 13, 17]`, indicate the status of the array after each iteration of BubbleSort and InsertSort.

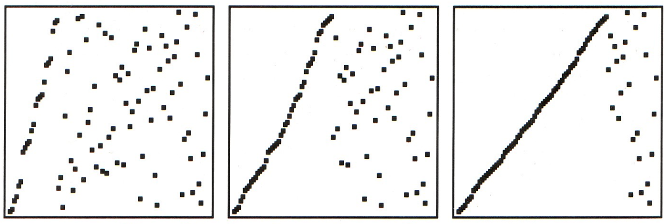
30. **(E)** Given the following list: `16 6 3 4 9 31 1 8 2 7`, explain the evolution of InsertSort reasonably indicating the status of the array after relocating each element. Indicate and count the number of key comparisons.

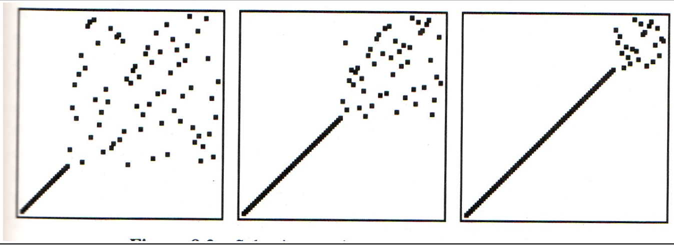
31. **(E)** Given the following list: `8 18 5 6 10 3 4 11 31 7`, explain the evolution of BubbleSort reasonably indicating for the first iteration what elements are being used as the bubble and detailing the partial status of the array. For the rest of iterations, just provide the status of the array after the iteration. How many iterations will the algorithm perform?

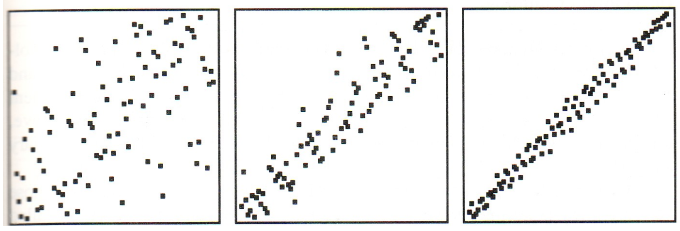
Problems

32. **(P; R)** Three consecutive phases in the evolution different sorting algorithms on an input array of numbers between 1 and 50 are displayed below. Coordinates (i, j) in these plots indicate that the number at index i is j (e.g., a number i is in the right location if $j = i$, i.e., if the pair (i, j) is in the diagonal of the plot). What sorting methods studied during the course correspond to each plot?









5 Runtime of local search algorithms

Exercises

33. **(E; R)** Given the following permutations of size 6 σ_6 (6 1 5 2 4 3), (6 5 4 1 2 3) y (2 3 6 4 5 1), calculate their transposes, count the number of inversions for both the original permutation and the transpose and check that they all add up the same number.

    If a local sorting number is used to sort them, what is the minimum number of key comparisons that the algorithm will perform?

34. **(E; R)** What is the minimum number of key comparisons that a local sorting algorithm will perform on the following permutation of $N = 2K$ elements? `(2K 2K-1 2K-2 ... K+1 1 2 3 ... K)`?

35. **(E; R)** What is the minimum number of key comparisons as a function of N that a local sorting algorithm will perform on the following permutation of $N = 3K$ elements?

    `(2K+1, 2K+2, ..., 3K, 2K, 2K-1, ... K+2, K+1, 1, 2, ..., K)`

36. **(E)** What is the minimum number of key comparisons as a function of N that a local sorting algorithm will perform on the following permutation of $N = 4K$ elements?

`(3K+1, 3K+2, ..., 4K, 3K, 3K-1, ..., 2K+2, 2K+1, 2K, 2K-1, ...  K+2, K+1, 1, 2, ..., K)`

37. **(E; R)** What is the minimum number of key comparisons as a function of N that a local sorting algorithm will perform on the following permutation of $N = 3K$ elements?

$$(\mathtt{3K, 3K-1, , ..., 2K+1, K+1, K+2, ..., 2K, K, K-1, ..., 2, 1}).$$

Problems

38. **(P)** Proof using the definition of transpose permutation that for a given permutation σ, $(\sigma^\tau)^\tau = \sigma$.

39. **(E+; R)** If a local sorting algorithm A is applied on the following permutation of $N = 2K$ elements: `(1 2K 2 2K-1 3 2K-2 ...  K K+1)`. How many key comparison will A perform on such permutation?

40. **(E+; R)**

    What is the minimum number of key comparisons as a function of N that a local sorting algorithm will perform on the following permutation of $N = 2K$ elements? `(2K, 1, 2K-1, 2, ..., 2K-i, i+1, ..., K+1, K)`.

41. **(P)** Calculate $W_{BubbleSort}(N)$, $A_{BubbleSort}(N)$, $W_{SelectSort}(N)$ and $A_{SelectSort}(N)$.

42. **(E)** We define $B_{\mathcal{A}}(N)$ as $B_{\mathcal{A}}(N) = \min\{n_{\mathcal{A}}(I) : I \in E_N^{\mathcal{A}}\}$. Analyze $B_{InsertSort}(N)$, $B_{SelecSort}(N)$, $B_{BubbleSort}(N)$.

43. **(E+)** Modify the pseudocode of InsertSort so that it can work with any given F and L indexes.

44. **(P; R)** Analyze $B_{InsertSort}(N)$, $B_{SelecSort}(N)$, $B_{BubbleSort}(N)$ using swaps as basic operation. Analyze also $W_{SelectSort}(N)$ and $W_{BubbleSort}(N)$ from this perspective.

45. **(P+; R)** Provide an estimation of $A_{BubbleSort}(N)$ considering the following pseudocode:

```
Bubble(array T, dim N)
  switch = 1;
  long   = N;
  while switch == 1 and long > 1:
    switch = 0;
    for i=1 to long-1:
      if T[i] > T[i+1]:
        swap(T[i],T[i+1]);
        switch = 1;
    long--;
```

46. **(P; R)** The following pseudocode is an alternative for the InserSort algorithm:

```
InsertSort(array T, dim N)
  for i=2 to N:
    j=i;
    while j > 1 and T[j-1] > T[j]:
      swap(T[j-1],T[j]);
      j--;
  return;
```

Using the swap as basic operation, reasonably calculate $W_{InserSort}(N)$ and $A_{InserSort}(N)$.

47. **(P)** A sorting algorithm is said to be **stable** if when applying it to an array with elements that may be repeated, two elements with the same key (and thus relatively already sorted) keep their relative positions at the end of the algorithm. Which of the studied sorting algorithms are stable (modify their pseudocode if needed?

6 MergeSort and QuickSort

Exercises

48. **(E)** Considering the following permutations:

    (a) (5 9 8 2 3 10 1 7 4 6),

    (b) (3 10 1 7 5 9 8 2 4 6),

    (c) (10 1 7 3 9 8 2 5 4 6),

    describe the evolution of Quicksort's `Divide` function assuming that its subroutine `Mid` returns index `P`.

49. **(E)** Describe the evolution of Quicksort's `Divide` on the array `(7 8 2 4 5 3 10 1 6)`.

50. **(E+)** Considering the list `[18, 7, 4, 6, 12, 23, 2]`,
    a. Indicate using a tree the distinct sublists generated in the Divide subroutine when MergeSort is used to sort the above list (place the sublist in a node and the corresponding next sublists on the left and on the right of such node). Number each node according to the order in which MergeSort generates the corresponding list.
    b. Considering the elemental sublists of the bottom level of the tree, indicate with an inverted tree the distinct sublists generated by MergeSort's Combine routine (e.g., the last node should contain the initial list already sorted). Number each node here according to the order in which MergeSort generates its corresponding list.

51. **(E)** Considering the lists `[45 29 47 32 19 12 26 51]` and `[12 9 22 37 25 32 6 14]`, study the evolution on a binary tree when MergeSort and Quicksort algorithms run on them.

52. **(E)** Considering the lists
    `[Q B K C F A G P] , [34 43 12 58 17 92 21]`
    indicate the sorting evolution on a binary tree when MergeSort runs on them.

Problems

53. **(P+)** Let us assume that in a table element `T[i]` is in inversion with element `T[i+K]`. Proof that if we swap them, we solve at least one inversion and at most $2K-1$ inversions.

54. **(P+)** Estimate the abstract runtime of QuickSort in the best and in the worst cases considering the swap between elements as the basic operation.

55. **(P)** Calculate $A_{Divide}(N)$, where *Divide* is the routine that QuickSort uses to divide the table in two subtables. Use the element swap as the basic operation.

56. **(P+)** Calculate the average abstract runtime of QuickSort using the element swap as the basic operation.

57. **(P)** Calculate $B_{Combine}$ y $B_{MergeSort}(N)$.

58. **(P+)** Calculate $A_{Combine}$ when the routine operates on two tables of size N.

59. **(P+)** Estimate the value of $A_{MergeSort}(N)$.

60. **(P+)** An alternative basic operation in MergeSort could be the "copy" of the elements that `Combine` implements. Estimate the values of $W_{MergeSort}(N)$ and $A_{MS}(N)$ using this choice of basic operation.

61. **(P+)** Analyze $B_{QuickSort}(N)$.

62. **(P)** We would like to apply the QuickSort algorithm taking always P as pivot on a set of arrays T of size $N = 2M+1$ in the form of

$$[M+1, 2, \ldots M, 1, T[M+2], \ldots, T[N]],$$

where T[j] values , $M+2 <$ j $\leq N$ are between $M+2$ and $N = 2M+1$; i.e., the first part of the table is always the same, in the form T[1] = M+1, T[i] = i if 1 < i < $M+1$, T[M+1]=1 and the second part varies, with unsorted integers from $M+2$ to N. Reasonably estimate the average case of such sorting as a function of N.

7 Recurrent inequalities

Exercises

63. **(E; R)** Express as a function of N the result of the following summations:

$$\begin{array}{llllll}
1) & \sum_0^{K-1} 2^j, \quad N = 4^K; & 2) & \sum_0^{K-j} 4^j, \quad N = 2^K; & 3) & \sum_0^{K-1} 2^j, \quad N = 3^K; \\
4) & \sum_0^{K-1} 2^{j+2}, \quad N = 8^K; & 5) & \sum_0^{K-1} \frac{3^{j-2}}{4^{j+2}}, \quad N = 12^K; & 6) & \sum_0^{K-1} 2^{j/2}, \quad N = 4^K; \\
7) & \sum_0^{K-1} \left(\frac{2}{3}\right)^j, \quad N = \left(\frac{3}{2}\right)^K; & 8) & \sum_0^{K-1} 4^j, \quad N = 2^K; & 9) & \sum_0^{K-1} 2^j, \quad \lg N = 3^K
\end{array}$$

64. **(E; R)** We know with regard to a certain function T that $T(1) = 0$ and $T(N) \leq \lg N + T(\lfloor N/2 \rfloor)$. Estimate the order of $T(N)$ for N in the form of $N = 2^K$ (Hint: iterate the inequality).

65. **(E; R)** Estimate the order of function $T(N)$ in the following recurrent inequalities assuming that $T(1) = 0$

    (a) $T(N) \leq 2T(\lfloor N/2 \rfloor) + N^3$;

    (b) $T(N) \leq 2T(N-1) + 1$;

    (c) $T(N) \leq T(N-1) + N$;

    (d) $T(N) \leq 2T(\lfloor N/4 \rfloor) + \sqrt{N}$;

66. **(E; R)** Reasonably estimate the growth of a function T that satisfies $T(1) = 0$, and the following recurrent inequality

$$T(N) \leq \sqrt{N} + 4T(\lfloor N/4 \rfloor).$$

67. **(E; R)**

    Estimate the growth of a function $T(N)$ with an integer argument that follows the recurrent inequality $T(N) \leq N + 2T(\lfloor N/2 \rfloor)$, where $T(1) = 0$.

Problems

68. **(P; R)**

    Reasonably estimate the growth of a function T that satisfies $T(1) = 0$, and the following recurrent inequality

$$T(N) \leq T(\lfloor N/2 \rfloor) + N \lg(N).$$

69. **(P; R)** Estimate the order of function $T(N)$ when $T(2) = 0$ and $T(N) \leq T(\left\lfloor \sqrt{N} \right\rfloor) + 1$.

8 Analysis of recurrent algorithms

Exercises

70. **(E; R)** The following algorithm solves the Tower of Hanoi puzzle to move N disks from a rod A to another rod B using a third rod as an intermediate store.

```
Hanoi(disks N, rod A, rod B, rod C)
  if N == 1:
    move one disk from A to B
  else
    Hanoi(N-1, A, C, B)
    move one disk from A to B
    Hanoi(N-1, C, B, A)
```

    Using the movement of disks as BO, how many movements will be made to transfer N disks from a to B? How many recursive calls will the algorithm run for N disks?

Problems

71. **(P; R)** The following algorithm returns the index of the maximum element between positions F and L in an array:

```
ind MaxRec(array T, ind F, ind L)
   si F == L:
       return F
   M = (F+L)/2
   m1 = MaxRec(T, F, M)
   m2 = MaxRec(T, M+1, L)
   if T[m1] > T[m2]:
       return m1
   else:
       return m2
```

Using the key comparison as basic operation, reasonably estimate the runtime in the worst case of the algorithm for arrays with N elements. For this task, write a recurrent inequality, solve it for a special case and, finally, provide the solution for all N.

72. **(P; R)** The following pseudocode corresponds to a recursive version for the calculation of the mean of an array:

```
float mean(array T, pos F, pos L)
   si F == L :
      return T[F]
   else :
      M = floor ((F+L)/2)
      meanI = mean(T, F, M)
      meanD = mean(T, M+1, L)
      nL = M-F+1         // # elem. in the left subarray
      nR = L-M           // # elem. in the right subarray
      nT = L-F+1         // Total # of elements
      mea = ( nL*meanL + nR*meanR ) / nT
      devolver mea
```

Assuming that the sum is the basic operation, estimate the number of sums $n(N)$ that the algorithm performs on an array of N elements following these steps:
i. write a recurrent equation for $n(N)$;
ii. solve it for a convenient special case;
iii. solve it or narrow the solution for the general case.

73. **(P)** Assuming that $N = 2^K$, a number A of N figures can be decomposed as $A = A_1 * D_K + A_2$, where A_1, A_2 have $N/2$ figures and $D_K = 10^{2^{K-1}}$. If we decompose two numbers A, B of N figures in this way, the formula

$$AB = A_1B_1 * D_K^2 + (A_1B_2 + A_2B_1) * D_K + A_2B_2$$

suggests the following recursive algorithm to multiply A times B:

```
int MMR(int A, int B, int N)
  if N==1:
    return A*B
  else:
    calculate A1, A2, B1, B2
    O = MMR(A1, B1)
    P = MMR(A1, B2) + MMR(A2, B1)
    Q = MMR(A2, B2)
    return (O * DK + P) * DK + Q
```

where `DK` is $10^{2^{K-1}}$. Considering the multiplication as the basic operation, what is the complexity $T_{MMR}(N)$ of running MMR on two numbers with $N = 2^K$ figures?

74. **(P+)** In the conditions of the previous problem, formula:

$$AB = A_1B_1 * D_K^2 + ((A_1 + A_2)(B_1 + B_2) - A_1B_1 - A_2B_2) * D_K + A_2B_2$$

reduces the multiplication of two numbers of N figures to three multiplications of two numbers of $N/2$ figures. Write the pseudocode of a recursive algorithm $MMRmR$ to multiply two numbers which uses such formula and estimate its complexity $T_{MMRmR}(N)$ when having two numbers of $N = 2^K$ figures as inputs.

75. **(P+; R)**

We would like to estimate the number of recursive calls of the following Tower of Hanoi algorithm in which the tail recursion has been eliminated. Write the corresponding recurrent equation and solve it.

```
hanoi2(int N, int A, int B, int C)
   while N > 1 :
      hanoi(N-1, A, C, B)
      move from A to B
      N--
      T = A ; A = C ; C = T
   move from A to B
```

76. **(P+; R)** The following pseudocode calculates the number of leaves in a binary tree:

```
int numLeaves(bt T)
   if T empty:
      return 0
   else if left(T) empty and right(T) empty:
      devolver 1
   else:
      return numLeaves(left(T)) + numLeaves(right(T)) ;
```

i. Reasonably write the general recurrent equation for the number of sums that this algorithm performs as a function of the number of nodes `N`.

ii. Solve the equation for complete binary trees (if such a tree has height K its number of nodes is s $2^K - 1$).

iii. What is the algorithm's worst case abstract runtime for general binary trees? Hint: try to establish in what situation we could assume maximum work for the algorithm.

77. **(P+; R)** We would like to estimate the number of recursive calls for the following algorithm of preorder traversals on complete binary trees. Write the corresponding recursive equation and solve it.

```
PO(BT T)
   if T != NULL:
      visit(T)
      PO(left(T))
      PO(right(T))
```

78. **(P+)** We have eliminated the tail recursion in the following version of the previous algorithm of preorder traversals. Estimate the number of recursive calls for complete binary trees. First write the recursive equation and then solve it.

```
PO2(BT T)
   while T != NULL:
      visit(T)
      PO2(left(T))
      T = right(T)
```

79. **(P+)** The following recursive algorithm calculates the Fibonacci numbers:

```
long Fib(int N)
   if N == 0 or N == 1:
      return 1
   else :
      return Fib(N-1) + Fib(N-2)
```

If $T(N)$ is the number of sums that `Fib` executes on an integer N, What is the recursive equation for $T(N)$?

What is the solution of this equation?

80. **(P+; R)** What can be say about $G(N) = \sum_0^N F_i$, where F_i is the i-t Fibonacci number?

9 HeapSort

Exercises

81. **(E)** Given the following arrays
`Q B K C F A G P E D H R`
`34 43 12 58 17 92 21 67 56 72 80`
1. build their corresponding Max heaps indicating the evolution of this construction in each step;
2. use these heaps to sort them, indicating the status at each step.

82. **(E)** Apply HeapSort to sort the array `8 6 2 10 9 1 3 5 7 4`.

83. **(E)** Consider the array: `[14, 4, 8, 10, 19, 17, 18, 15]`. Apply HeapSort using a max-heap. Indicate the partial status of CreatHeap on a binary tree representation. During SortHeap, represent the status of the heap and of the corresponding array at each step.

84. **(E; R)** Consider the array: `[15, 1, 10, 5, 8, 3, 11, 14]`, Write the evolution of HeapSort indicating each step in the construction of the maxheap and the corresponding sorting process.

85. **(E)** Write the evolution of HeapSort for `9 8 10 2 5 6 7 12 11`.

Problems

86. **(P)**

Assuming that the basic operation is the access to a node, what would be the complexity in the worst case when creating a maxpheap? What would be the complexity in the worst case for sorting the maxheap?

Combining both steps we get a comparison based sorting algorithm. What is its complexity in the worst case?

87. **(P+)** Reasonably estimate $B_{HeapSort}(N)$ and $A_{HeapSort}(N)$

88. **(P)**

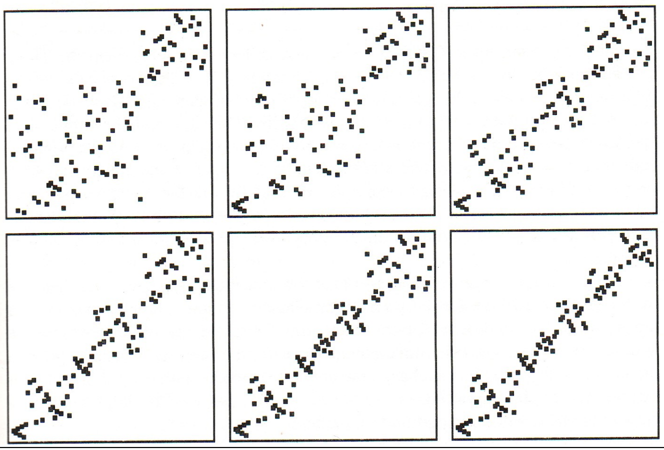
The elements of a heap can be easily stored in an array by making that the element that is located in the i–th position of the heap (when it is read from top top to bottom and from left to right) is in the i–th position of the array. What relationship exists between the index i of a node and its children? What is the relationship between the index i and the parent node?

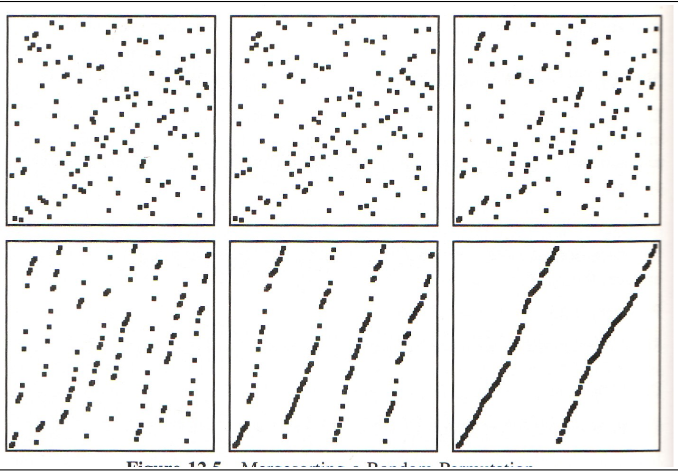
89. **(P) Implementation of Heapsort on arrays** If according to the previous problem we represent a table with indexes from 1 to N as a heap, we can observe that the only elements that cannot meet the MaxHeap conditions are the ones located in indexes

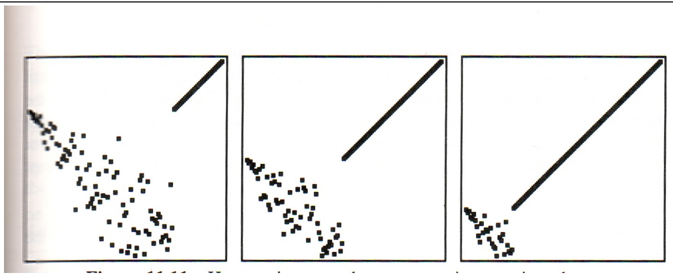
$\lfloor N/2 \rfloor, \lfloor N/2 \rfloor - 1, \lfloor N/2 \rfloor - 2, \ldots, 3, 2, 1.$
Show that relocating those elements in that order we can get a MaxHeap.

90. **(P)** Once a MaxHeap is created in an array, show that the array can be sorted by repeatedly swapping the elements in positions $N, N-1, \ldots$ with the root element and relocating the element that "climbs up" that way.

91. **(P; R)** In the following diagrams, three consecutive phases of certain sorting methods are shown when applied on an array with 50 numbers between 1 and 50. The coordinates (i, j) in a diagram indicate that the number in the i-th position of the array is j (e.g., that a number i is in the right position if $j = i$, i.e., if the pair (i, j) is in the diagonal). What sorting methods analyzed during the course could generate these diagrams?







10 Decision Trees

Exercises

92. **(E)** Write the desion tree for BubbleSort, InsertSort and SelectSort for $N = 3$.

93. **(E; R)** Write the decision tree for QuickSort and MergeSort for $N = 3$

94. **(E)** A sorting algorithm has been designed with the following pseudocode:

```
RareSort(array T, ind F, ind L)
   if T[F] < T[F+1]:
      InsertSort(T, F, L) ;
   else:
      MergeSort(T, F, L) ;
```

    Write its decision tree for 3 element arrays.

95. **(E)** Reasonably write the decision tree for $N = 3$ of the following sorting algorithm. Use notation `i:j` with $i < j$ to indicate the comparison of elements i and j in the initial array `T`.

```
RareSort(tabla T, dim N)
   if T[1] < T[2]:
      QuickSort(T, N) ; // use the first element as pivot
   else :
      InsertSort(T, N) ;
```

96. **(E+)** Reasonably write the decision tree for $N = 3$ of the following variant of the `SelectSort` algorithm:

```
SelectSortMax(array T, dim N)
  for i=N to 2:
    iMax=Max(T, 1, i);
    swap T[i] with T[iMax];

index Max(array T, index F, index L)
  Max=T[F];
  iMax=F;
  for i=F+1 to L:
    if T[i] > Max:
       Max = T[i];
       iMax=i;
  return iMax;
```

97. **(E+; R)**

    The InsertSort algorithm may be described by the following alternative pseudocode:

```
InsertSortMax(array T, dim N)
  for i=N-1 to 1:
    A = T[i];
    j = i+1;
    while j <= N and T[j] < A:
      T[j-1] = T[j];
      j++;
    T[j-1] = A;
```

Reasonably write its decision tree for $N = 3$ indicating after each key comparison the status of the array.

98. **(E+; R)** Reasonably write the decision tree for input arrays of size 3 for the following version of the bubble sort algorithm:

```
LeadSort(array T, dim N)
  change = 1;
  end = 1;
  while end < N y change == 1:
    change = 0;
    for i=N To ini+1:
      If T[i] < T[i-1]:
        swap T[i] and T[i-1];
        change = 1;
    end++;
```

Write also the decision tree after removing the lines with the `change` flag.

99. **(E)** a. Given an array T, the InsertSort algorithm performs first the comparison between $T[1]$ and $T[2]$. Assuming that $T[1] > T[2]$, what is the element $T[j]$ against which $T[1]$ is then compared?

    b. Assuming that $T[1]$ wins such comparison, reasonably write the decision subtree that follows such key comparison for arrays with 4 elements (i.e., the subtree that follows on the right of the root is $1 : 2$, and its right child is in the form $1 : j$). Indicate after each key comparison the status of the array and mark the element to be inserted.

Problems

100. **(E+; R)** Given an array of four elements, the MergeSort algorithm performs first the comparison `1:2` and if the first element is larger, then the algorithm performs the comparison `3:4`. Write the rest of the decision tree assuming that the first element is larger than the second and that the third element is less than the fourth element. Indicate after each step the status of the array.

101. **(E+; R)** The following pseudocode is a "descending" version of the BubbleSort algorithm in which the array is sorted starting from the left.

```
LS(array T, dim N)
  k = 1;
  while k < N :
    for i=N to k+1:
      if T[i] < T[i-1]:
        swap T[i] and T[i-1];
    k++;
```

In an array of 4 elements `LS` performs first the comparison 3 : 4. If $T[3] > T[4]$, what is the next key comparison? Assuming that in the following key comparison the element with the larger index is less than the other element, reasonably write the resulting decision subtree in T^4_{LS}.

102. **(E+; R)** The following algorithm is a "descending" version of InserSort:

```
DIS(array T, dim N)
  for i=N-1 to 1 :
    j=i ;
    while j < N and T[j] > T[j+1] :
      swap(T[j], T[j+1]) ;
      j++ ;
  return;
```

In an array of 4 elements `DIS` performs first the comparison 3 : 4. If $T[3] > T[4]$, what is the next key comparison? Assuming that in the following key comparison the element with the larger index is less than the other element, reasonably write the resulting decision subtree in T^4_{DIS}.

103. **(E+; R)** Write the decision tree for the HeapSort algorithm for 3-element arrays.

104. **(P; R)** The building of a binary search tree (BST) can be seen as a particular case of sorting on arrays. What would be its decision tree for $N = 3$?

105. **(P; R)** Assuming that the building of a BST is applied to arrays with $N = 4$ elements, the first key comparison is `1:2` and the second 1:3. Reasonably write the decision tree for $T^4_{CreateBST}$ assuming also that `T[1] > T[2]` and `T[1] < T[3]`. Represent the resulting partial BSTs after each key comparison .

106. **(P; R)** Proof that if T is a 2-tree with N nodes then T has $N + 1$ leaves (hint: induction).

107. **(P)** Establish in a precise manner the relationship between the height of a binary tree and its number of leaves.

108. **(P)** If a complete binary tree has N elements, what is its height? what is its leave path length (LPL)?

109. **(P+; R)** The leave path length (LPL) of a tree T is defined as

$$LPL(T) = \sum_{\{N:\text{internal node of } T\}} height(N).$$

Proof that if T is a 2-tree with N internal nodes, then $LPL(T) = LPL(T) + 2N$ (hint: induction).

11 Basic search algorithms

Exercises

110. **(E)** Given an array T

     ```
     1 15 2 14 3 13 4 12 5 11
     ```

     write its binary search tree B.
     Write the binary search trees that result from extracting elements `15, 13, 3` from B in that order.

111. **(E+)** Calculate $A_{BBin}(N)$ for $N = 3$ and $N = 7$ (assume equiprobability).

Problems

112. **(P)** Estimate the average cost of `Remove` in a dictionary that uses as a data structure a sorted array.

113. **(P; R)**

     Prove that if a node of a BST has two children, its successor has a single right child at most.

114. **(P)** If `N` is a node in a BST, its predecessor is the node `M` such that `info(M)` is the key immediately before to `info(N)`. Proof that if a node of a BST hast two children, its predecessor has at most a left child.

115. **(P+; R)** Write the pseudocode of a function `Successor` that receives a pointer to a node of a binary tree and returns the pointer to is successor.

116. **(P+)** Write the code of a function `Predecessor` that receives a pointer to a node of a binary tree and returns the pointer to its predecessor.

12 AVL Trees

Exercises

117. **(E)** Build the AVL tree associated to the array `1 2 3 4 5 6 7 15 14 13 12 11 10 9 8`

118. **(E)** Build the AVL tree associated to the reverse of the previous array (from 15 to 8 and from 1 a 7), indicating at each step in detail the necessary rotations.

119. **(E)**

     Build the AVL tree associated to the array `1 15 2 14 3 13 4 12 5 11 6 10 7 9 8`.

120. **(E; R)** Build the AVL tree associated to the array `9 8 10 2 5 6 7 12 11`.

121. **(E; R)** Build all Fibonacci trees with height 2, 3 y 4 (there are not that many!).

Problems

122. **(P)** Write the general scheme of a right rotation and of a double left-right rotation.

123. **(P+)** Prove that in the process of creating an AVL no unbalance of type (2,0) or (0,2) can appear.

124. **(P+)** Prove that in the process of creating an AVL it is enough to fix the deepest unbalance and the upper ones are automatically fixed.

125. **(P+)** A **Fibonacci tree** of height P is an AVL tree with such height and a minimum number of nodes. Show that if T is a Fibonacci tree with height P, then T_l is a Fibonacci tree of height $P-1$ and T_r is a Fibonacci tree of height $P-2$, or vice versa, T_l is a Fibonacci tree of height $P-2$ and y T_r is a Fibonacci tree of height $P-1$.

126. **(P+)**

     If we suppose that rabbits become adults in one month and that a pair or adult rabbits, independent of their sex, produce a pair of baby rabbits in a month. If two young adults arrive to a desert island, how many pairs of rabbits will be in the island after N months? (e.g., after two months there will be two pairs: one of baby rabbits and one of adult rabbits).

127. **(P+; R)**

     Prove by induction that the n-th Fibonacci number F_N meets that

$$F_N = \frac{1}{\sqrt{5}}(\phi^{N+1} - \psi^{N+1})$$

     with $\phi = (1+\sqrt{5})/2$ and $\psi = (1-\sqrt{5})/2$. Find from this relationship the order of function $f(N) = F_N$.

128. **(P+; R)** Calculate the order of $G(N) = \sum_0^N F_i$, where F_i is the i-th Fibonacci number.

129. **(P+; R)** How many Fibonacci trees of height P exist?

13 Hashing

Exercises

130. **(E)**

     Build an adequate hash function for a hash table with chaining and adequate size to store the following values: 5, 28, 19, 15, 20, 33, 12, 17 and 10.

131. **(E; R)** We would like to build a hash table with chaining to store 1000 data. If an average of 4 probes is considered acceptable for both successful and unsuccessful searches, what would be an adequate size for the table?

132. **(E; R)**

We would like to build a hash table with open addressing and linear probing to store 1000 data. If an average of 5 probes is considered acceptable in unsuccessful searches, what would be an adequate size for the table?

For the resulting table size, what would be the average number of probes in successful searches?

133. **(E+; R)**

It can be shown that for quadratic probing in a hash table of dimension m and N data the average number of probes needed in a failed search is:

$$A^f_{QP}(N,m) \approx \frac{1}{1-\lambda} - \lambda + \log\frac{1}{1-\lambda},$$

where λ is the load factor. What would be the average number of probes in successful searches $A^s_{QP}(N,m)$?

134. **(E+; R)** In a certain method of hash with **chaining** HX we know that $A^f_{HX}(N,m) = \lambda^2$. Reasonably estimate the value of $A^s_{HX}(N,m)$.

135. **(E+)** In a certain method of hash with open addressing PX we know $A^f_{PX}(N,m) = 1+1/(1-\lambda)^2$. Reasonably estimate the value of $A^s_{PX}(N,m)$.

Problems

136. **(P; R)** In general, regarding hash tables, what would be larger, the average cost of failed searches or the average cost of successful searchers. Provide an explanation for the answer.

137. **(P+; R)** We would like to build dynamic hash tables with open addressing and linear probing according to the following scheme named **rehashing**: when in an m-sized table the load factor reaches 0.5, we build a new table of size $2m$ where elements are reinserted from the previous table.

On average, how many probes are needed to build a table with m elements? What is the extra cost of this procedure as compared to building a table with $2m$ positions for $N = m$ data?

138. **(P; R)** If in a hash table with open addressing we use quadratic probing to solve collisions, there can be keys impossible to insert even though the table has free positions. Test this effect when the table has size 16 and the hash function $h(K) = K\%16$ is used.

139. **(P+; R)**

Prove that if in a hash table of size m with m prime, open addressing, load factor $\lambda < 1/2$, and quadratic probing is used to solve collisions, it is always possible to insert a new key k.

Hint: assume P elements already inserted, with $P < \lfloor m/2 \rfloor$ and check that given a key k, for any hash function h, all values of the set

$$\Big\{(h(k)+i^2)\%m \;:\; 0 \le i \le \lfloor M/2 \rfloor\Big\}$$

are different; note that in such set there are more than P elements.

140. **(P)**

   Write the pseudocode of routines that search, insert and remove elements in a hash table with open addressing and linear probing to solve collisions.

141. **(P)**

   In the context of a hash function with chaining, we have built a routine `SearchH(key K, tableH T)` that receives a key `K` and returns a pointer to a node of the list pointed by `T[h(K)]`, which contains such key or `NULL` if no node contains it.

   Build using this routine the routines `DeleteH(key K, tableH T)` and `InsH(key K, tableH T)` which remove or insert key `K` from or at the appropriate node of the hash table `T`. Build with this latter routine another routine `CreateH(table R, tableH T)` such that it provides from a table `R` a pointer to a hash table that contains the elements of `R`.