HONORS PROJECTS – optional

Basically you can take one of the algorithmic ideas discussed in class and implement it. The choice of test cases and documentation is left largely to you. Here are some ideas. If you have one of your own I'll be glad to listen. Also if you do work on them I can further explain the ideas.

Merge and Find

Begin with n vertices $1, \ldots, n$ and no edges and add edges at random, using Merge and Find to keep track of the component size. Let your output be the size L of the largest component as a function of the number of edges e in the graph. If you do it right you'll find a surprising behavior near $e = \frac{n}{2}$ when L will suddenly grow. It turns out that a good parametrization is $e = \frac{n}{2} + \lambda \frac{n^{3/2}}{2}$ and $L = Mn^{2/3}$. For several values of n (with a good program even $n = 10^6$ will run fairly quickly) run the algorithm until $\lambda = +3$. For $\lambda = -3, -2, -1, 0, 1, 2, 3$ find M. Run this many times for each n and give the seven histograms for M. You should get the same histograms for the different n.

HeapSort

Implement heapsort on n random uniformly chosen numbers. That is, first build the heap on n elements and then `EXTRACTMIN` n times to get the sorted array. During the `EXTRACTMIN` phase keep track of the number of times (call it X) that an element is exchanged with its child. (This occurs when you `HEAPIFY-DOWN` form the root.) Run the algorithm many times for each of several values of n. For each value of n give a histogram for the values X. Try to come up with some reasonable conjectures about how X behaves as a function of n. (In class we noted that $X \leq n \lg n$ since each `HEAPIFY-DOWN` has at most $\lg n$ exchanges but this is only an upper bound.)

TicTacToe

We can use DFS to solve Tic-Tac-Toe. Create the graph with vertices the possible positions of a Tic-Tac-Toe board. (As an upper bound there are $3^9 \sim 20000$ such because each spot is X,O or blank but also the number of X is either the same or one more than the number of O. (X goes first.) Make it a directed graph by letting $Adj(v)$ be the set of positions you can get to from v with the next move. Let Z denote the empty position, at the start of the game. Add a new field called $value(v)$, which has values W, L, D depending on whether the position is a win,loss or draw for the player about to move.

The idea is to apply $DFS(v)$ with some modifications. First: when doing $DFS(v)$ first check whether the "previous" player just won the game. In that case set $value(v) \leftarrow L$, color v black, and return. Also, if all nine moves have been made and there is no winner set $value(v) \leftarrow D$, color v black, and return. Second: after going through all the neighbors of v in $DFS(v)$: set $value(v) = W$ if any $value(w) = L, w \in Adj(v)$. Set $value(v) = L$ if all $value(w) = W, w \in Adj(v)$. Set $value(v) = D$ otherwise (i.e., no $value(w) = L$ but some $value(w) = D$. (This makes sense. E.g., if there is any way to put your opponent in a losing position then you have a winning position.)

You should output $value(Z)$ (TicTacToe is a draw!) and any other values that your 10-year old nephews and nieces find interesting.

Quicktimes

We will identify numbers with arrays, one digit in each position, so that 492 is identified with the array with $A(0) = 2, A(1) = 9, A(2) = 4$. We always start counting at zero, from the right. $QUICKTIMES(A, B, D)$ will take two arrays A, B and give an array D for their product. We will use two auxilliary programs $ADD(x, y, z)$ and $SUBTRACT(x, y, z)$ which, with input array x, y give array z with $z = x + y$ and $z = x - y$ respectively. Here the "usual" programs work in time $O(n)$. We split A into its left and right halves, call them a, b. For example, with $A = 49302841$ we have $a = 4930$ and $b = 2841$. We similarly split B into c, d. Thus, if each number has n digits.

$$A = 10^{n/2}a + b \text{ and } B = 10^{n/2}c + d$$

We apply recursively

$$QUICKTIMES(a, c, e) \text{ so that } e = ac$$

$$QUICKTIMES(b, d, f) \text{ so that } f = bd$$

Now for the key step. We apply

$$ADD(a, b, g) \text{ and } ADD(c, d, h)$$

so that

$$g = a + b \text{ and } h = c + d$$

and then

$$QUICKTIMES(g, h, i)$$

so that

$$i = gh = (a + b)(c + d) = ac + bd + (bc + ad)$$

We apply

$$SUBTRACT(i, e, j)\text{and then } SUBTRACT(j, f, k)$$

so that

$$k = j - f = i - e - f = (a + b)(c + d) - ac - bd = ad + bc$$

The number we want is $AB = 10^n e + 10^{n/2}k + f$ and so that is found with two more ADDs.

The project would be to take two numbers of, say, $32,000$ digits each and multiply them together. A nice aspect would be to also program the "usual" multiplication and compare how long they take.