B1 Algorithms (25 points)

Select any 5 of the following 6 items (do only 5 -- if you do all 6, only the first 5 will be graded). For each item that you select, give a short explanation of the item (data structure, algorithm, or technique). You should capture the main idea/ideas of the data structure, algorithm, or technique in your explanation, and, for algorithms, state the running time.

1) Prim's algorithm

2) The linear time (worst case) algorithm for selection

3) Dynamic programming

4) The Floyd/Warshall algorithm

5) Heapsort

6) FFT

B2 Algorithms (25 points)

Do all three parts: a, b and c.

In this question *YOU* are asked to design some web site management tools. For each of them, discuss your design strategy. What data structures and algorithms will you use? What will be the time and memory cost functions of your solutions? Assume the following basic functions are available to use:

1. URL* anchors_list(URL X)
2. boolean good_page(URL X) returns true if and only if X references a valid page.
3. boolean same_site(URL A, URL B) returns true if and only if A and B are part of the same site.
4. boolean same_URL(URL A, URL B) returns true if and only if A and B reference the same page.

Parts a and b that follow are "essay questions". It is important that you use your answers as a vehicle to show your knowledge of the relevant data structures and algorithms. Please discuss the properties of them as well as your use of them in the design of the two functions. It may also help to mention data structures considered but not chosen for the task and explain why.

a) [12 points] **Dead Link Finder**

Design the function DLF(). Here, given URL A, DLF(A) must return a list of URL's which are reachable from A, but do not reference valid web pages. You may use the existing functions from the list above. In particular, anchors_list(URL X) returns a list of the URL's linked on the page X. DLF(A) is expected to return URL's of pages linked directly from page A or indirectly through links from A to other pages on the same site.

Note that there may be circular references among the pages. You may assume that there are no more than 10000 pages on a given site. Measure the time and memory cost in terms of the numbers m and/or n. Here m is the number of pages on the same site as A and reachable directly or indirectly from A, and n is the number of pages directly linked to those.

b) [13 points] **How far my reach?**

Design the function Extent(URL A). This function returns the count of all the web pages anywhere in the world reachable from page A. You can assume the URL's of the reachable pages fit on disk but not on main memory. Clever ideas to reduce the number of stored URL's will be appreciated (and given more credit). Remember to analyze the time and memory cost functions of your solutions. Clearly explain the parameter(s) you use for these functions.

B3 Algorithms (25 points)

Do both part a and part b.

a. **Binomial Heaps**

i. [3 points] What is a binomial heap?

ii. [6 points] Explain how to perform a union operation on two binomial heaps. Also state the running time of such a union, assuming that each heap contains n elements.

b. **Fibonacci Heaps**

i. [2 points] What is a Fibonacci Heap? (ie explain the data structure)

ii. [2 points] How are Fibonacci Heaps related to binomial heaps?

iii. [3 points] What is meant by an amortized running time?

iv. [3 points] Explain the implementation of a decrease key operation in a Fibonacci Heap, and state its running time (you do not have to prove its running time).

v. [6 points] Give an example of an algorithm that uses Fibonacci Heaps -- explain how Fibonacci Heaps are used there, and how they figure in the running time of the algorithm.

B4 Algorithms (25 points)

Do all five parts.

a. [3 points] Define NP-complete. You may assume that P, NP and polynomially reducible have already been defined.

b. [3 points] Explain how to prove that a problem is NP-complete. Be sure to mention all of the critical elements.

c. [4 points] Name and briefly describe four *different* NP-complete problems (the four problems should NOT be variations of the same problem, nor should any of them involve cliques, PARTITION or KNAPSACK).

d. [7 points] One of the classic NP-complete problems is CLIQUE and yet we know that the following problem (5CLIQUE) is solvable in polynomial time: Given a graph G, determine whether or not G contains a clique of size 5. Explain how this can be, and also give the running time of a polynomial algorithm for solving 5CLIQUE.

e. [8 points] It is well-known that PARTITION is NP-complete (ie given $x_1, x_2, \ldots, x_n$, does there exist A, a subset of $x_1, x_2, \ldots, x_n$, such the sum of the numbers in A is equal to the sum of the numbers not in A?).

A related problem is KNAPSACK. Here you are given numbers $z_1, z_2, \ldots, z_m$ and a value B and are asked: Does there exist Q, a subset of $z_1, z_2, \ldots, z_m$ such that the sum of the numbers in Q is equal to B?

Give a reduction from PARTITION to KNAPSACK. You do not have to prove that the construction that you give is correct. Just give the construction itself.