Análisis de Algoritmos y Matemáticas Discretas
Tarea Heapsort

Antonio Barragan Romero

Problema 1

1. Nuestros criterios de comparacion pues nos interesa ver que $A[\text{parent}(i)] \leq A[i]$ dado que queremos formar un min-heap.
2. Primero supongamos que A está ordenado de mayor a menor, no se cumple la propiedad del min-heap, es identico al caso siguiente.
   Ahora veamso que pasa cuando A está ordenado de menor a mayor. En este caso A ya es un min-heap, sin embargo cuando hacemos el swap intercambiamos el elemento mas grande con el mas pequeños invalidando la propiedad del min-heap, Entonces al hacer el min-heapify debemos recorrer todo el arbol $(\log(n))$ pues el elemento más grande debe ser una hoja, por lo cual el tiempo de ejecución es $n\log(n)$.
3. El tiempo de ejecución es independiente de si el heap es max o min.

Problema 2

Se adjunta codigo heapsort.cpp.

Problema 3

Primero Debemos ver que la invariante se mantiene antes del ciclo:

Inicialización Antes del ciclo convertimos A en un heap[1], entonces cuando $i = n$ se cumple que $A[1 : n]$ es un heap que contiene los n elementos mas pequeños de A y por vacuidad se cumple que $A[n + 1 : n]$ contiene los 0 elemntos mas grandes de A ordenados.

Despues debemos ver que cada iteración mantiene la invariante:

Mantenimiento La invariante nos dice que $A[1 : i]$ es un heap con los i elementos mas pequeños de A, entonces $A[1]$ es el elemento mas grande(mas pequeño) de $A[1 : i]$, por lo cual al hacer el swap se cumple que $A[1 : i-1]$ tiene los $i-1$ elementos más pequeños de A y además $A[i : n]$ tiene los $n-(i-1)$ elementos más grandes de A en orden pues por hipotesis $A[i + 1 : n]$ son los elementos mas grandes de A en orden, despues del swap tenemos que $A[i]$ es el elemento mas grande de $A[1 : i]$ los cuales son menores o iguales a $A[i + 1]$, por lo tanto $A[i : n]$ esta en orden. Hasta este momento $A[i : n]$ tiene los $n-(i-1)$ elementos mas grandes de A en orden y $A[1 : i-1]$ los $i-1$ elementos mas pequeños, la linea max-heapify nos asegura que $A[1 : i-1]$ es un max heap y por tanto se cumple la inveriante para la siguiente ejecución del ciclo $(i-1)$.

Por ultimo veamos que pasa cuando el ciclo termina.

[1] Un max-heap

Finalización El ciclo termina cuando $i = 2$, por lo anterior tenemos que despues del swap y el max-heapify se cumple que $A[1 : 1]$ es un heap con los 1 elementos mas pequeños de A y que $A[2 : n]$ contiene los $n - 1$ elementos mas grande de A en orden y por tanto podemos concluir que $A[1 : n]$ esta ordenado, como queremos.

Problema 4

1. Dado que tenemos todos los niveles del arbol llenos excepto posiblemente el ultimo entonces, como tiene altura h entonces $h - 1$ niveles estan llenos, por lo cual el arbol tiene al menos $\sum_{i=0}^{h-1} 2^i = 2^h - 1$ nodos, debe tener h niveles es decir al menos un nodo en el nivel h y a lo mas 2^h nodos, se sigue que tiene al menos $2^h - 1 + 1 = 2^h$ y a lo mas $2^h - 1 + 2^h = 2^{h+1} - 1$
2. Si, pues al estar ordenado cumple que $A[\text{parent}(i)] \leq A[i]$ para todo i dado que parent(i) <= i.
3. No es un max-heap pues parent(9)=4 y $16 = A[9] > A[4] = 15$ (1-based-index)