University of Zurich[UZH]

Department of Informatics
Database Technology Group
Prof. Dr. M. Böhlen

Data Structures and Algorithms
Assignment 3

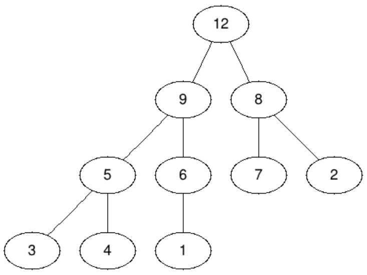
Mar 21, 2016

Heap and Heapsort

Task 1.

Output:

```
graph g {
  12 -- 9
  12 -- 8
  9 -- 5
  9 -- 6
  8 -- 7
  8 -- 2
  5 -- 3
  5 -- 4
  6 -- 1
}
```



```
#include <stdio.h>

#define MAX_ARRAY_SIZE 100

int parent(int i) {
   return (i - 1) / 2;
}

int lchild(int i) {
   return (i + 1) * 2 - 1;
}

int rchild(int i) {
   return (i + 1) * 2;
}

void heapify(int A[], int i, int s) {
   int l = lchild(i);
   int r = rchild(i);
   int max = i;
```

```
    if ( l < s && A[l] > A[max] ) max = l;
    if ( r < s && A[r] > A[max] ) max = r;

    if ( max != i ) {
        /* Swap */
        int tmp = A[i];
        A[i] = A[max];
        A[max] = tmp;

        heapify(A, max, s);
    }
}

void printArray(int a[], int n) {
    int i;

    printf("[ ");
    for(i = 0; i < n; i++) {
        printf("%d", a[i]);
        if(i < n-1) {
            printf(", ");
        }
    }
    printf(" ]\n");
}

void buildMaxHeap(int A[], int n) {
    int i;

    for (i = n / 2; i >= 0; i--)
        heapify(A, i, n);
}

void printHeap(int A[], int n) {
    int i, l, r;

    printf("graph g {\n");
    for(i = 0; i < n; i++) {
        l = lchild(i);
        r = rchild(i);
        if(l < n) printf(" %d -- %d\n", A[i], A[l]);
        if(r < n) printf(" %d -- %d\n", A[i], A[r]);
    }
    printf("}\n");
}

void heapSort(int A[], int n) {
    int i;

    for (i = n - 1; i > 0; i--) {
        int tmp = A[i];
        A[i] = A[0];
        A[0] = tmp;
        n--;
        heapify(A, 0, n);
    }
}

void main() {
    int a[MAX_ARRAY_SIZE];
    int i, n;

    printf("Type elements of A seperated by spaces (type 'end' to stop)
        : ");
    i=0;
    while(scanf("%d", &a[i]) == 1) i++;
    n=i;
    // Read but do not store any terminating not integer values ('end')
    scanf("%*s");

    buildMaxHeap(a, n);
    printHeap(a, n);

    heapSort(a, n);
    printf("Sorted: ");
    printArray(a, n);
}
```

Quicksort

Task 2

```
#include <stdio.h>
#include <math.h>

#define MAX_ARRAY_SIZE 100

void swap(int A[], int i, int j) {
     int tmp = A[i];
     A[i] = A[j];
     A[j] = tmp;
}

void printArray(int a[], int size) {
     printf("[");
     int i;
     for(i = 0; i < size; i++) {
          printf(" %d", a[i]);
     }
     printf(" ]\n");
}

int medianOfThree(int A[], int low, int high) {
     int m = (low + high) / 2;

     if(A[m] < A[low]) swap(A, m, low);
     if(A[low] > A[high]) swap(A, high, low);
     if(A[m] > A[high]) swap(A, high, m);

     return m;
}

int partitionHoare(int A[], int low, int high) {
     int pivot= A[high], i=low-1, j= high+1;

     while(1) {
          while(A[--j] > pivot);
          while(A[++i] < pivot);

          if(i < j) swap(A, i, j);
          else return i;
     }
}


int quicksort(int A[], int low, int high, int choice) {
    int m;

    if(low < high) {
        if(choice == 1) {
             m = medianOfThree(A, low, high);
             swap(A, high, m);
        }
        m = partitionHoare(A, low, high);
        quicksort(A, low, m-1, choice);
        quicksort(A, m , high, choice);
    }
}

void main() {
    int a[MAX_ARRAY_SIZE];
    int i, n;
    int choice=1;

    printf("Type elements of A seperated by spaces (type 'end' to stop)
         : ");
    i=0;
    while(scanf("%d", &a[i]) == 1) i++;
    n=i;
    // Read but do not store any terminating not integer values ('end')
    scanf("%*s");

    quicksort(a, 0, n-1, 0);
    printf("Sorted: ");
    printArray(a, n);
}
```

Task 3

```
#include <stdio.h>
#include <time.h>

#define MAX_ARRAY_SIZE 120000

// ------------------------------ heap sort ---------------------------------

int parent(int i) {
    return (i - 1) / 2;
}

int lchild(int i) {
    return (i + 1) * 2 - 1;
}

int rchild(int i) {
    return (i + 1) * 2;
}

void heapify(int A[], int i, int s) {
    int l = lchild(i);
    int r = rchild(i);
    int max = i;

    if ( l < s && A[l] > A[max] ) max = l;
    if ( r < s && A[r] > A[max] ) max = r;

    if ( max != i ) {
        /* Swap */
        int tmp = A[i];
        A[i] = A[max];
        A[max] = tmp;

        heapify(A, max, s);
    }
}

void buildMaxHeap(int A[], int n) {
    int i;

    for (i = n / 2; i >= 0; i--)
        heapify(A, i, n);
}

void heapSort(int A[], int n) {
    int i;

    for (i = n - 1; i > 0; i--) {
        int tmp = A[i];
        A[i] = A[0];
        A[0] = tmp;
        n--;
        heapify(A, 0, n);
    }
}

// ----------------------------- quick sort ---------------------------------

void swap(int A[], int i, int j) {
    int tmp = A[i];

    A[i] = A[j];
    A[j] = tmp;
}

int medianOfThree(int A[], int low, int high) {
     int m = (low + high) / 2;

     if(A[m] < A[low]) swap(A, m, low);
     if(A[low] > A[high]) swap(A, high, low);
     if(A[m] > A[high]) swap(A, high, m);

     return m;
}

int partitionHoare(int A[], int low, int high) {
    int pivot=A[high];
    int i=low-1;
    int j= high+1;

    while(1) {
        while(A[--j] > pivot);
        while(A[++i] < pivot);

        if(i < j) swap(A, i, j);
        else return i;
    }
}

int quicksort(int A[], int low, int high, int choice) {
    int m;

    if(low < high) {
        if(choice == 1) {
             m = medianOfThree(A, low, high);
             swap(A, high, m);
        }
        m = partitionHoare(A, low, high);
        quicksort(A, low, m-1, choice);
        quicksort(A, m , high, choice);
    }
}

// ---------------------------- bubble sort ---------------------------------

void bubblesort(int a[], int size) {
    int t, i, j;
    int cnt=0;
    for (i = size - 1; i > 0; i--) {
        for (j = 1; j <= i; j++) {
            if (a[j] < a[j-1]) {
                cnt++;
                t = a[j];
                a[j] = a[j-1];
                a[j-1] = t;
            }
        }
    }
}

// ------------------------- auxiliary functions ------------------------------

void readFile(char filename[], int output[], int *n) {
    FILE *f;
    int i;

    f=fopen(filename, "r");

    i=0;
    while(fscanf(f, "%d", &output[i]) == 1) i++;
    *n=i;

    fclose(f);
}

// ------------------------------- main ---------------------------------------

void main() {
    int a[MAX_ARRAY_SIZE];
    int n;
    int choice;
    clock_t start;
    clock_t end;
    float seconds;

    printf(" | ordered | inverse | random |\n");

    // --------------------------- heap sort ----------------------------------

    // ordered
    readFile("ordered.txt", a, &n);
    start = clock();
    heapSort(a, n);
    end = clock();
    seconds = (float)(end-start)/CLOCKS_PER_SEC;
    printf("heapsort | %7.4f | ", seconds);

    // inverse
    readFile("inverse.txt", a, &n);
    start = clock();
```

```
heapSort(a, n);
end = clock();
seconds = (float)(end-start)/CLOCKS_PER_SEC;
printf("%7.4f |", seconds);

// random
readFile("random.txt", a, &n);
start = clock();
heapSort(a, n);
end = clock();
seconds = (float)(end-start)/CLOCKS_PER_SEC;
printf(" %7.4f |\n", seconds);

// ------------------------- quick sort ---------------------------------

choice = 0;
// ordered
readFile("ordered.txt", a, &n);
start = clock();
quicksort(a, 0, n, choice);
end = clock();
seconds = (float)(end-start)/CLOCKS_PER_SEC;
printf("quicksort (simple) | %7.4f | ", seconds);

// inverse
readFile("inverse.txt", a, &n);
start = clock();
quicksort(a, 0, n, choice);
end = clock();
seconds = (float)(end-start)/CLOCKS_PER_SEC;
printf("%7.4f |", seconds);

// random
readFile("random.txt", a, &n);
start = clock();
quicksort(a, 0, n, choice);
end = clock();
seconds = (float)(end-start)/CLOCKS_PER_SEC;
printf(" %7.4f |\n", seconds);

// ------------------------- quick sort ---------------------------------

choice = 1;
// ordered
readFile("ordered.txt", a, &n);
start = clock();
quicksort(a, 0, n, choice);
end = clock();
seconds = (float)(end-start)/CLOCKS_PER_SEC;
printf("quicksort (median of 3) | %7.4f | ", seconds);

// inverse
readFile("inverse.txt", a, &n);
start = clock();
```

```
    quicksort(a, 0, n, choice);
    end = clock();
    seconds = (float)(end-start)/CLOCKS_PER_SEC;
    printf("%7.4f |", seconds);

    // random
    readFile("random.txt", a, &n);
    start = clock();
    quicksort(a, 0, n, choice);
    end = clock();
    seconds = (float)(end-start)/CLOCKS_PER_SEC;
    printf(" %7.4f |\n", seconds);

    // ------------------------ bubble sort ---------------------------------

    // ordered
    readFile("ordered.txt", a, &n);
    start = clock();
    bubblesort(a, n);
    end = clock();
    seconds = (float)(end-start)/CLOCKS_PER_SEC;
    printf("bubblesort | %7.4f | ", seconds);

    // inverse
    readFile("inverse.txt", a, &n);
    start = clock();
    bubblesort(a, n);
    end = clock();
    seconds = (float)(end-start)/CLOCKS_PER_SEC;
    printf("%7.4f |", seconds);

    // random
    readFile("random.txt", a, &n);
    start = clock();
    bubblesort(a, n);
    end = clock();
    seconds = (float)(end-start)/CLOCKS_PER_SEC;
    printf(" %7.4f |\n", seconds);
}
```

Results:

	ordered	inverse	random
heapsort	0.0016	0.0327	0.0121
quicksort (simple)	15.9597	15.9346	0.0194
quicksort (median of 3)	0.0077	0.0127	0.0206
bubblesort	19.4663	34.6706	44.6433