Programming Assignment 4: Sorting Algorithms

Yuqin Ao

November 8, 2025

Abstract

This report presents the implementation and testing of five sorting-related algorithms: MAX-HEAPIFY, BUILD-MAX-HEAP, Heapsort, Counting Sort, and Radix Sort. All implementations were written in Python, and test results are included to demonstrate correctness.

1 Implementation and Results

Maxheapify

```
#max heapify
class myheap:
    def __init__(self,value):
        self.value=value
        self.length=len(value)
        self.heapsize=len(value)
    def  __str__(self):
        return str(self.value)
def Maxheapify(A,i):
    l=2*i
    r=2*i+1
    if (l<=A.heapsize) and (A.value[l-1]>A.value[i-1]):
        largest=l
    else:
        largest=i
    if (r<=A.heapsize) and (A.value[r-1]>A.value[largest-1]):
        largest=r
    if largest!=i:
        A.value[i-1], A.value[largest-1] = A.value[largest-1], A.value[i-1]
        Maxheapify(A,largest)
    return A
```

test

```
import math
B=[16,4,10,14,9,3,2,8,1]
H=myheap(B)

MaxH=Maxheapify(H,2)

print(MaxH)
```

Output:

```
[16, 14, 10, 8, 9, 3, 2, 4, 1]
```

build max heap

```
#build max heap
def build_maxheap(A):
    for i in range(A.length//2,0,-1):
        Maxheapify(A,i)
    return A
```

test

```
B=[4,1,3,2,16,9,10,14,8,7]
H=myheap(B)
MaxH=build_maxheap(H)
print(MaxH)
```

Output:

```
[16, 14, 10, 8, 7, 9, 3, 2, 4, 1]
```

heapsort

```
#heapsort
def heapsort(A):
    for i in range(A.length,1,-1):
        A.value[i-1], A.value[0] = A.value[0], A.value[i-1]
        A.heapsize-=1
        Maxheapify(A,1)
    return A
```

test

```
B=[16, 14, 10, 8, 7, 9, 3, 2, 4, 1]
H=myheap(B)
MaxHS=heapsort(H)
print(MaxHS)
```

Output:

```
[1, 2, 3, 4, 7, 8, 9, 10, 14, 16]
```

radix sort(include counting sort)

```
import numpy as np
#count sort
def countsort(A,EXP):
    C=np.zeros(10,dtype=int)
```

```
    n=len(A)
    B=np.zeros(n,dtype=int)
    digits=(A//EXP)%10
    for i in digits:
        #index=(i//EXP)%10
        C[i]+=1
    for j in range (1,10):
        C[j]+=C[j-1]
    for m in range (n-1,-1,-1):
        B[C[digits[m]]-1]=A[m]
        C[digits[m]]-=1
    return B
#radix sort
def radixsort_core(A):
    if not isinstance(A, np.ndarray):
        A = np.array(A, dtype=int)
    maxV=np.max(A)
    exp=1
    while maxV//exp>0:
        A=countsort(A,exp)
        exp*=10   ####
    return A
def radixsort_neg(A,max_decimal):
    neg = [-x for x in A if x < 0]
    pos = [x for x in A if x >= 0]

    neg_sorted = radixsort_core(neg)[::-1] if len(neg)>0 else []
    pos_sorted = radixsort_core(pos) if len(pos)>0 else []
    A=[-x for x in neg_sorted] + list(pos_sorted)
    J=0
    for i in A:
        A[J]=i/10**max_decimal
        J+=1
    return A
def radixsort(A):
    A=np.array(A,dtype=float)
    neg_inf=np.isneginf(A)
    pos_inf=np.isposinf(A)
    finite=np.isfinite(A)
    neg_inf_v=A[neg_inf]
    pos_inf_v=A[pos_inf]
    A=A[finite]
    max_decimal=0
    for num in A:
        if (num % 1)!=0:
            decimal=str(num).split('.')[1]
            max_decimal=max(max_decimal,len(decimal))
    if max_decimal>0:
        A*=10**max_decimal
        print(A)
    A_sorted=np.concatenate([neg_inf_v,radixsort_neg(A,max_decimal),
       pos_inf_v])
    return A_sorted
```

test

```
A = [300, 324, 311, 325, 308, 324]
print(radixsort(A))
B=[329,457,657,839,436,720,355]
print(radixsort(B))
C = [329, 457, 89,436, 63, 7]
print(radixsort(C))
```

Output:

```
[300. 308. 311. 324. 324. 325.]
[329. 355. 436. 457. 657. 720. 839.]
[  7.  63.  89. 329. 436. 457.]
```

2 Conclusion

In this assignment, all required algorithms were successfully implemented and tested. The output results match the expected behavior described in the lecture examples.